TSSI - Arquitectura y sistemas operativos

Conceptos y diseño de Sistemas Operativos

Ing. Alejandro Mostovoi

Año 2023

Revisión: 1.9

Tabla de versiones

Autor	Fecha	Versión	Descripción
Alejandro Mostovoi	15/08/22	1.0	Cambios en Interrupciones
Alejandro Mostovoi	23/08/22	1.1	Unidad II. Agrega Intro y definición de procesos.
Alejandro Mostovoi	29/08/22	1.2	Unidad III. Mejora definiciones, gráficos. ULT y KLT
Alejandro Mostovoi	04/10/22	1.3	Mejoras en definiciones de técnicas de tratamiento de interbloqueo.
Alejandro Mostovoi	10/10/22	1.4	Cambios menores, índice, gramática.
Alejandro Mostovoi	30/10/22	1.5	Buddy System - Recupero de memoria. Memoria requerida por un proceso
Alejandro Mostovoi	08/11/22	1.6	Agrega link Wikipedia - RAID
Alejandro Mostovoi	09/11/23	1.7	Corrección ventajas de RAID 1
Alejandro Mostovoi	15/05/24	1.8	Corrección sinc. por mensajes.
Alejandro Mostovoi	13/06/24	1.9	Agrega información sobre DMA y DMAC

Índice

Introducción a los sistemas operativos **2**

¿Qué es un sistema operativo? 2

El sistema operativo como una máquina extendida 2

El sistema operativo como un administrador de recursos 3

Objetivos y funciones 4

Facilidad de uso 5

Gestionar los recursos 8

Capacidad para evolucionar 8

Un poco de historia y algo más 8

Procesamiento en serie (primera generación) 9

Sistemas en lotes simple (segunda generación) 10

Sistema en lotes multiprogramado (tercera generación) 11

Sistema de tiempo compartido 13

Conceptos Básicos 14

El procesador 14

Registros visibles para el usuario 14

Registros de control y estado 14

El shell 15

Ciclo de ejecución de instrucciones 16

System Calls 16

Interrupciones 16

Ciclo de instrucción con interrupciones 17

Técnicas de comunicación de E/S 18

E/S programada 18

E/S por interrupciones 18

DMA 19

Los 5 pilares de un sistema operativo 19

Procesos 19

Gestión de almacenamiento 21

Archivos 21

Memoria Virtual 22

Protección y seguridad de la información 23

Planificación y gestión de los recursos 23

Estructura del sistema 24

Diseño de un sistema operativo 25
Monolíticos 25
En capas 26
Microkernels 26
Cliente-Servidor 27
Máquina Virtual 27
Exokernel 29
Procesos 29
Introducción 29
Definición 30
Descripción y control de procesos 31
Estados de un proceso 31
Modelo de 2 estados 31
Modelo de 5 estados 33
El estado Nuevo 34
Modelo de 7 estados 35
Estructuras de control de procesos 36
Control de procesos 37
Modos de ejecución 37
Cambio de contexto (context switch) 38
Interrupción 38
Excepción 38
Llamada al sistema (system call) 40
Cambio de modo 40
Ejecución del sistema operativo 41
Núcleo sin procesos 41
Ejecución dentro de los procesos de usuario 41
Sistemas operativos basados en procesos 42
Procesos e Hilos 42
Introducción 42
Definición 43
Multihilo 44
Diferencias entre proceso e hilo 46
Ventajas de usar hilos 47
Estados de los hilos 47
Sincronización de hilos 48
Hilos en el espacio de usuario o en el espacio del kernel 48
Hilos en el espacio de usuario o ULT 49

Hilos en el espacio del kernel o KLT 51
Concurrencia, exclusión mutua y sincronización 51
Introducción 51
Principios de la concurrencia 52
Interacción entre procesos 54
Procesos que no se perciben entre sí - Competencia 54
Procesos que se perciben indirectamente - Cooperación por compartición 56
Procesos que se perciben directamente - Cooperación por comunicación 57
Exclusión mútua 57
Soporte por Hardware 57
Técnicas por software 58
Semáforos 58
Monitores 58
Paso de mensajes 59
Fundamentos del interbloqueo (deadlock) 61
Recurso Reutilizable 62
Recurso Consumible 62
Condiciones para el interbloqueo 62
Prevención del interbloqueo 63
Métodos indirectos 63
Exclusión mutua 64
Retención y Espera 65
Sin expropiación (no preemption) 65
Métodos directos 66
Espera circular 66
Evitar el interbloqueo 66
Denegación de inicio de un proceso 67
Denegación de asignación de un recurso (algoritmo del banquero) 68
Detección y recuperación del interbloqueo 70
Recuperación del interbloqueo 72
Planificación del procesador 73
Tipos de planificación 73
Planificación a largo plazo 73
Planificación a medio plazo 74
Planificación a corto plazo 74
Relación entre planificador y los estados de un proceso 74
Algoritmos de planificación 74
Criterios 74

Políticas 77
Primero en llegar, primero en servirse (FCFS) 77
Round Robin 79
Primero el proceso más corto (SPN) 79
Menor tiempo restante (SRT) 80
Feedback 80
Gestión de la memoria **81**
Requisitos 81
Reubicación 81
Protección 82
Compartición 82
Organización lógica y Organización física 82
Particionamiento 82
Estático (fijo) 82
Particiones de igual tamaño 82
Particiones de tamaño variable 84
Dinámico (variable) 85
Algoritmo de ubicación 86
Sistema Buddy 87
Técnicas avanzadas de gestión de memoria 88
Paginación 89
Segmentación 90
Memoria Virtual **91**
Introducción 91
Proximidad y memoria virtual 94
Paginación 95
Esquemas de Tabla de páginas 97
Dos Niveles 97
Tabla de páginas invertida 98
Buffer de traducción anticipada (TLB) 99
Tamaño de la página 99
Paginación y segmentación 100
Políticas del sistema operativo 100
Políticas de recuperación 101
Políticas de ubicación 101
Políticas de reemplazo 102
Algoritmos básicos 102
Óptimo 102

LRU 102
FIFO 103
Reloj 103
Reloj mejorado 104
Buffering de páginas 104
Gestión del conjunto residente 105
Políticas de limpieza 106
Control de carga 107
Gestión de Entrada Salida 108
Introducción 108
Evolución del sistema de E/S 108
DMA 109
¿Cómo funciona el acceso directo a memoria (DMA)? 109
Uso de buffers 112
Buffer único 114
Buffer doble 114
Buffer circular 115
Planificación de disco 115
Políticas de planificación 116
FIFO 116
Prioridad 116
SSTF 116
SCAN 117
C-SCAN 117
SCAN-N y FSCAN 117
Cache de disco 117
Consideraciones de diseño 118
RAID 119
RAID 0 120
RAID 1 120
RAID 2 120
RAID 3 121
RAID 4 121
RAID 5 121
RAID 6 121
Bibliografía 121

Unidad I

~ Introducción a los sistemas operativos ~

En el principio...fue la línea de comandos

~ Neal Stephenson[1]

[1] https://es.wikipedia.org/wiki/En_el_principio_fue_la_l%C3%ADnea_de_comandos

Introducción a los sistemas operativos

¿Qué es un sistema operativo?

Ante todo, un sistema operativo es un programa que permite controlar la ejecución de aplicaciones, que actúa como interfaz entre las aplicaciones y el hardware de la computadora.

Es complejo dar una definición precisa y parte de esta complejidad se debe a que un sistema operativo realiza dos funciones que en principio no se encuentran relacionadas entre sí. La primera de estas funciones es extender la funcionalidad de la máquina mientras que la segunda se refiere a la administración de los recursos de la máquina.

El sistema operativo como una máquina extendida

Desde el punto de vista de un programador/a, resultaría muy complejo tener que desarrollar software utilizando las instrucciones en lenguaje máquina de cada arquitectura de computadoras. Imaginemos que si por cada programa a desarrollar debemos implementar las rutinas para lectura o escritura de datos en disco o memoria, la programación se tornaría una actividad cuanto menos tediosa, sobre todo si tenemos en cuenta que estas instrucciones son dependientes de la arquitectura y por lo tanto cambian máquina a máquina.

Es por esto que una función del sistema operativo es presentar al usuario el equivalente de una **máquina extendida** o **máquina virtual** de forma tal que sea más fácil de programar que el hardware subyacente.

La forma en que el sistema operativo logra esto es proporcionando una variedad de servicios que los programas pueden obtener usando instrucciones especiales denominadas **llamadas al sistema**.

El sistema operativo como un administrador de recursos

El concepto anterior puede entenderse como una visión “top-down”. Una visión alternativa, “bottom-up” (de abajo hacia arriba), sostiene que el sistema operativo está para administrar todas las piezas de un sistema complejo.

Las computadoras modernas constan de procesadores, memorias, discos, interfaces de red, dispositivos de E / S y una amplia variedad de otros dispositivos. La función del sistema operativo es proporcionar una asignación ordenada y controlada de estos componentes entre los diversos programas que compiten por el uso de los mismos

La administración de recursos, implica que los mismos deben ser compartidos por los distintos programas o usuarios que los solicitan de dos maneras: tiempo y espacio.

Cuando un recurso es compartido en el tiempo, significa que los procesos (o usuarios) que lo solicitan, deben usar dicho recurso por turnos. Un ejemplo de esto sería que varios procesos requieran utilizar el CPU al mismo tiempo. Es responsabilidad del sistema operativo asignar los turnos para que cada proceso o usuario pueda utilizar el recurso.

Por otra parte, recursos como la memoria o el disco, pueden ser fraccionados y asignados de a partes entre los procesos o datos que lo requieran. Asignar los espacios (direcciones) de memoria o los bloques de disco es responsabilidad del sistema operativo.

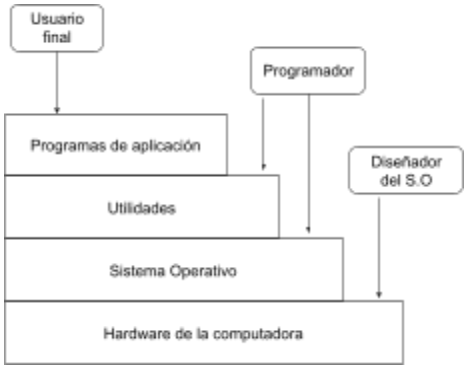
Objetivos y funciones

A grandes rasgos, un sistema operativo posee 3 objetivos que debe cumplir:

- Facilidad de uso
- Gestionar los recursos
- Capacidad para evolucionar

Facilidad de uso

El hardware y software utilizados para proporcionar aplicaciones a los usuarios se puede pensar como un sistema organizado jerárquicamente en capas, donde la capa inferior proporciona servicios a la capa superior.



Esta jerarquía de capas hace que un usuario final, que utiliza un programa de aplicación, no tenga que preocuparse por detalles del hardware. A su vez un programador de aplicaciones, para desarrollar una aplicación utiliza bibliotecas proporcionadas por el sistema operativo que le permiten controlar el hardware de la computadora. Imaginemos que sin estas bibliotecas cada

programador debería desarrollar todo un conjunto de instrucciones en código máquina que le permitan controlar la E/S (entrada/salida) de datos, manejo de archivos en disco, etc.

Finalmente el sistema operativo, resulta ser el software de base más importante ya que oculta los detalles del hardware a los programadores proporcionando una interfaz apropiada para su uso (bibliotecas) mientras que a las aplicaciones proporciona servicios y utilidades para realizar las tareas.

De manera resumida, un sistema operativo, proporciona servicios en las siguientes áreas:

- **Desarrollo y ejecución de programas**. Proporcionando bibliotecas y servicios al desarrollador. Realizando todos los pasos necesarios para ejecutar un programa.
- **Acceso a dispositivos de E/S**. Proporcionando un conjunto de instrucciones que pemita al programador y a las aplicaciones manipular de manera uniforme los dispositivos.
- **Acceso controlado a archivos**. Proporciona mecanismos de control y protección en el acceso.
- **Acceso al sistema**. Controla el acceso al sistema y a los recursos. Proporcionando protección a los recursos y datos.
- **Detección y respuesta a errores**. Ya sea por errores de hardware o software, debe proporcionar una respuesta que elimine la condición de error. La respuesta puede variar entre cerrar la aplicación que produjo el error hasta reintentar la operación.
- **Estadísticas**. Recolecta estadísticas de uso de los recursos y rendimiento. Estos datos son utilizados para mejorar el rendimiento del sistema.

Gestionar los recursos

La computadora es un conjunto de recursos (ej.: procesador, memoria, disco) que se utilizan para el ingreso, procesamiento y almacenamiento de datos así como también para llevar a cabo el control de estas operaciones. Ahora bien, el sistema operativo es el encargado de gestionar y controlar estos recursos.

Como ya mencionamos, el sistema operativo es un software que como tantos otros proporciona instrucciones al procesador de la máquina, pero lo que lo hace diferente es su objetivo. El sistema operativo dirige al procesador en el uso de los otros recursos de la computadora y administra su uso entre los distintos programas, es decir, el sistema operativo determina cuánto tiempo de procesador debe asignarse a la ejecución de un programa de aplicación/usuario.

Por otra parte, la memoria principal (RAM) es administrada en forma conjunta por el sistema operativo y el hardware de gestión de memoria del procesador.

Por último, es el sistema operativo el que debe decidir cuándo un programa puede utilizar un dispositivo de E/S (entrada / salida) a la vez que controla el acceso y uso de los archivos.

Capacidad para evolucionar

Un sistema operativo debe tener la capacidad de evolucionar en el tiempo debido a actualizaciones en el hardware y tipos de hardware, nuevos servicios o mejoras de los existentes y resolución de errores.

Un poco de historia y algo más

La primera computadora digital fue diseñada por el matemático inglés Charles Babbage (1792 - 1871). Esta máquina nunca llegó a construirse debido a que era 100% mecánica y en ese momento de la historia no contaba con la tecnología adecuada. Lo interesante de este episodio es que entre 1842 y 1843, *Ada Lovelace* comienza a trabajar con Babbage describiendo con un lenguaje técnico el funcionamiento de la máquina de Babbage. En su trabajo, incluyó varias notas entre las cuales figura una detallada y formal descripción de cómo obtener una secuencia de números de Bernoulli utilizando la máquina de Babbage. Gracias a esto, hoy es considerada la primera programadora de la historia.

El lenguaje de programación, Ada, fue nombrado así en honor a Ada Lovelace

Procesamiento en serie (primera generación)

Desde los años 40 y hasta mediados de la década del 50, el programador interactuaba directamente con el hardware de la computadora. No existía ningún sistema operativo. Los programas eran escritos en código máquina que se cargaba en la computadora a través de tarjetas perforadas en el dispositivo de entrada, un lector de tarjetas. Si el programa terminaba de forma normal, el resultado podía verse en la impresora, si se generaba un error, el programador podía examinar los registros del procesador y la memoria principal.

Este sistema tenía dos problemas. El primero era que la planificación se realizaba de forma manual. Esto implicaba reservar tiempo de procesamiento en una planilla con lo cual podía ocurrir que el programa terminara antes de lo estipulado, por lo que se desperdiciaba tiempo o bien que el programa tarde más de lo esperado y por lo tanto no se pudiera obtener los resultados.

El segundo problema era el tiempo de configuración. Cada vez que se tenía que ejecutar un programa, el programador debía cargar el compilador, el programa y los datos. Esto implicaba un tiempo considerable.

Se denominó *procesamiento en serie* debido a que los programas se ejecutaban uno atrás del otro.

Sistemas en lotes simple (segunda generación)

A mediados de los años 50, y con el surgimiento del transistor, las computadoras se volvieron más confiables y "aptas" para ser vendidas. Generalmente eran organismos gubernamentales y universidades quienes podían pagar las cifras multimillonarias.

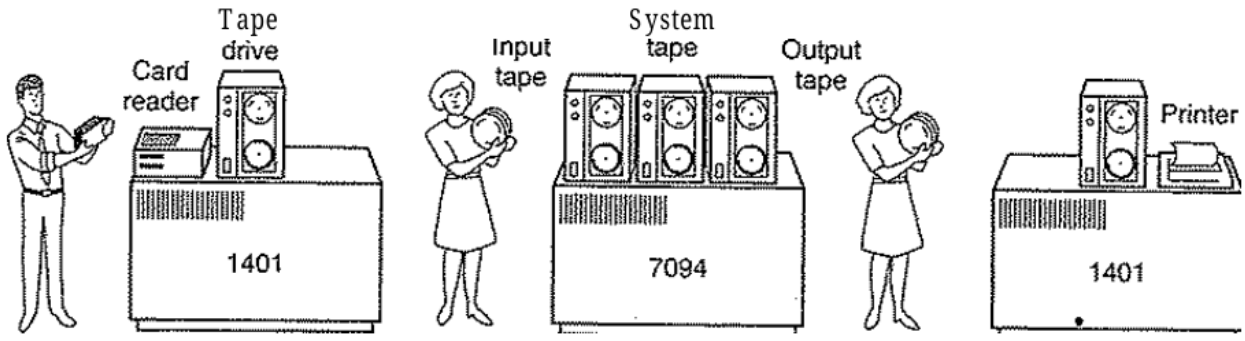
Estas máquinas, hoy conocidas como ***mainframes***, ocupaban grandes habitaciones que debían estar muy bien refrigeradas y debían ser operadas por personal capacitado.

Para salvar los problemas del procesamiento en serie y aprovechar mejor los tiempos, se desarrolló el concepto de sistema operativo en lotes. La idea central, es el uso de una pieza de software denominada **monitor** o **supervisor** que se encargaba de cargar y ejecutar los programas a medida que iban terminando.

De la misma manera que ocurría con los sistemas en serie, para ejecutar un programa (o trabajo), el programador debía enviar el mismo impreso en tarjetas perforadas al operador de la computadora, éste los organizaba y colocaba en el dispositivo de entrada, un lector de tarjetas.

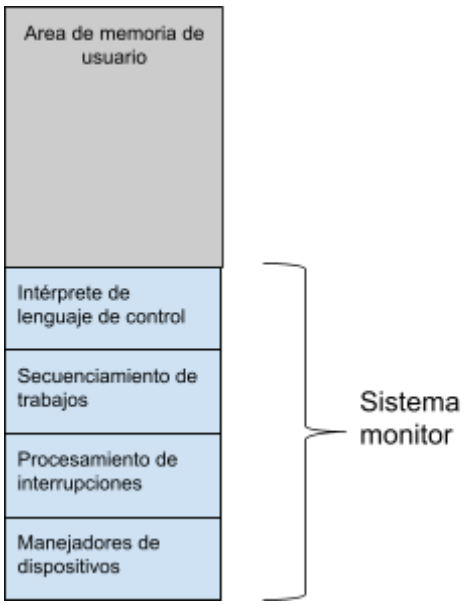
Para tratar de agilizar los tiempos, estos programas eran pasados de las tarjetas perforadas a una cinta magnética que luego era llevada por el operador a otra máquina. Esta segunda máquina es donde residía el ***sistema monitor***. Este sistema, podríamos decir el ancestro del sistema operativo moderno, era el encargado de cargar y ejecutar los programas de la cinta magnética. El resultado de cada programa era grabado en otra cinta magnética, que al finalizar todos los trabajos, el operador debía llevar a otra máquina que leía la cinta e imprimía los resultados para que los programadores pudieran analizarlos.

La ilustración siguiente resume los pasos mencionados:



Este nuevo esquema puede ser analizado desde 2 puntos de vista distintos, el del sistema monitor y el del procesador.

Monitor: es el encargado de controlar la secuencia de eventos. Para ello, gran parte del monitor debe estar siempre cargado en la memoria principal. A esta porción del monitor, se la llama ***residente***. El resto del monitor está formado por utilidades y funciones que se cargan a medida que los programas las requieren. El trabajo del monitor es leer cada uno de los trabajos de la cinta magnética y colocarlo en el área de usuario de la memoria principal. Una vez hecho esto, el monitor cede el control al programa. Una vez que el programa finalizó, el control es devuelto al monitor que cargará el siguiente trabajo.



Ahora bien, desde el punto de vista del procesador, el mismo sólo ejecuta instrucciones. Lo que sucede es que cuando el monitor carga un programa en la memoria de usuario, ejecuta una instrucción de salto que le indica al procesador que la próxima instrucción a ejecutar es la primer instrucción del programa cargado. De la misma forma, cuando el programa finaliza, el procesador encuentra una instrucción que le indica que la próxima instrucción a ejecutar es la siguiente del programa monitor.

Tener en cuenta, que el monitor no deja de ser un programa más, que delega responsabilidad en el procesador para cargar instrucciones de diferentes porciones de la memoria, que de forma alternativa le permiten ceder y recuperar el control.

Con el correr del tiempo, los fabricantes se dieron cuenta que había otras características del hardware que debían implementarse:

Protección de memoria: como mencionamos hay una porción de memoria destinada al sistema monitor y una a los programas de usuario. Este concepto apunta a que ningún programa de usuario modifique o acceda a la porción de memoria destinada al sistema monitor.

Instrucciones privilegiadas: hay instrucciones que pueden generar fallos y desestabilizar el sistema o que permiten acceder a recursos compartidos. Por ello estas instrucciones sólo pueden ser ejecutadas por el sistema monitor.

Interrupciones: esta es una característica muy avanzada, que se implementó mucho después del surgimiento del monitor. Otorga una mayor flexibilidad para ceder y retomar el control entre el sistema operativo y los programas de usuario.

Temporizador: se utiliza para evitar que un programa de usuario monopolice el sistema y evite que el resto de los programas se ejecuten. Cuando el temporizador expira se devuelve el control al sistema monitor.

Los conceptos de instrucciones privilegiadas y protección de memoria dieron origen a los modos de ejecución. Un programa de usuario ejecuta en **modo usuario**, mientras que el sistema operativo ejecuta en **modo núcleo (modo kernel)**.

Sistema en lotes multiprogramado (tercera generación)

A principio de los años 60, y con el surgimiento de los circuitos integrados, IBM desarrolló el equipo System/360 que básicamente integraba los 2 tipos de equipos que existían al momento, los orientados a cálculos matemáticos y los orientados a entrada y salida de datos, ordenamiento, etc. Los descendientes de estos equipos son ampliamente usados aún hoy en bancos y organizaciones con alto procesamiento de datos.

La innovación del System/360 era que tenía la capacidad de procesar varias tareas al mismo tiempo, lo que se dió a conocer como **multiprogramación** o **multitarea.**

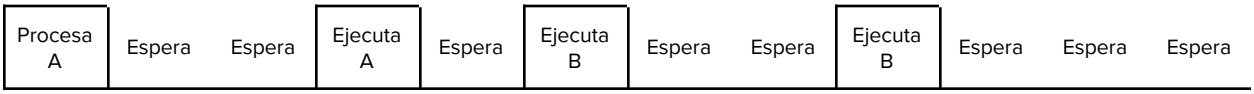
El principio detrás de este concepto es que el procesador se encuentra frecuentemente ocioso, por lo que se desaprovecha tiempo de procesamiento. Este tiempo ocioso, se debe principalmente a las operaciones de entrada / salida que generan los programas, lecturas y escrituras de datos desde y hacia discos (o cintas magnéticas), etc. Para tener una idea sobre este aspecto, consideremos el siguiente ejemplo:

Un programa realiza las siguientes operaciones: Leer un registro.....15 µs	Porcentaje de utilización de CPU: $\frac{1}{31} = 0,032 = 3,2\%$

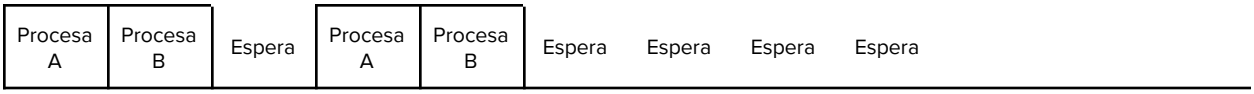
Ejecutar 100 operaciones.....1 µs Escribir un registro.....15 µs Total.....31 µs	Tiempo de CPU ocioso (aproximado): 96%

Bien, esta ineficiencia puede salvarse si consideramos que existe una cantidad de memoria principal tal que admita más de 1 programa cargado al mismo tiempo, además del sistema operativo (monitor residente). En este escenario, suponiendo que tenemos 2 programas cargados en la memoria, cuando el programa “A” realiza instrucciones de entrada/salida, el monitor puede asignar el control del procesador al programa “B” y viceversa.

Ejemplo:



a. sistema monoprogramado (lotes simples)



b. sistema multiprogramado

Al igual que los sistemas monoprogramados dependían de ciertas características del hardware para poder funcionar, los sistemas multiprogramados incorporan 2 características fundamentales para poder funcionar correctamente, que son:

- ❏ Interrupciones de E/S
- ❏ DMA: acceso directo a memoria

Las interrupciones de E/S, permiten que un controlador de un dispositivo de E/S interrumpa el accionar del procesador y que el control del mismo sea devuelto al programa que estaba esperando por esa operación de E/S.

DMA, permite a los controladores de dispositivos de E/S acceder directamente a la memoria principal sin tener que hacerlo a través del procesador.

Sistema de tiempo compartido

Con el advenimiento de los sistemas multiprogramados, el procesamiento se hizo cada vez más eficiente, pero en ciertos casos es deseable que el usuario pueda interactuar directamente con

la computadora, por ejemplo para realizar algún tipo de transacción. Es entonces cuando surge el concepto de sistemas de **tiempo compartido (time sharing).**

En los años 60, las computadoras personales no existían y para realizar trabajos interactivos los usuarios se conectaban desde una terminal o estación de trabajo a una computadora central que, a diferencia de las computadoras que conocemos hoy, ocupaba toda una habitación.

El concepto de *tiempo compartido*, como su nombre lo indica, comparte el uso de procesador entre los distintos usuarios conectados. Es un concepto similar a la multiprogramación pero que permite gestionar múltiples usuarios en vez de programas.

Estos sistemas entrelazan la ejecución de los programas de usuario utilizando pequeños intervalos de tiempo. Por lo tanto, si hay *n* usuarios activos solicitando un servicio a la vez, cada usuario sólo utilizará en promedio *1/n* de la capacidad de procesamiento de la computadora. Esta técnica debe su éxito a la lenta capacidad de reacción de los seres humanos en comparación con la velocidad en los tiempos de respuesta de las máquinas.

¿Cómo funciona un sistema de tiempo compartido?

Cuando el control se asigna a un usuario, el programa de usuario y los datos son cargados por el monitor en memoria principal. Un reloj del sistema genera una interrupción cada una determinada cantidad de segundos, en ese momento el sistema operativo retoma el control y asigna el procesador a otro usuario. Para preservar el estado del programa de usuario que es reemplazado, el programa y sus datos son escritos a disco antes de cargar el nuevo programa.

Para minimizar la escritura a disco, esta operación se realiza sólo en el caso de que el programa entrante requiera más memoria de la que el sistema tiene libre en ese momento.

Tanto la multiprogramación como los sistemas de tiempo compartido plantean nuevos desafíos. Al tener varios programas en memoria al mismo tiempo, se requieren técnicas para que un programa no lea/escriba la memoria de otro. Al tener múltiples usuarios el sistema de archivos debe ser protegido de forma que sólo los usuario autorizados tengan acceso a determinados archivos. Surgen los conflictos entre recursos como impresoras y dispositivos de almacenamiento.

Video recomendado:

- 1963 Timesharing: A Solution to Computer Bottlenecks: https://youtu.be/Q07PhW5sCEk

Conceptos Básicos

El procesador

El procesador, podríamos decir que es el “cerebro” de la computadora. Es quien controla el funcionamiento y realiza las funciones de procesamiento de datos.

Cada procesador posee su propio conjunto de instrucciones y es por esto que un Pentium no puede ejecutar un programa SPARC y viceversa.

El procesador tiene un conjunto de registros (una suerte de memoria interna) utilizados para guardar valores de variables y resultados intermedios de operaciones. Estos registros los podemos clasificar en:

Registros visibles para el usuario

Son registros que están disponibles para todos los programas de usuario y sistema. Son registros que normalmente se utilizan para almacenar datos, direcciones de memoria.

Registros de control y estado

Son registros que se utilizan para controlar el funcionamiento del procesador. El acceso a estos registros se realiza cuando se ejecuta en modo kernel. Algunos de los registros esenciales para la ejecución de programas son:

- PC (contador de programa): contiene la dirección de memoria de la próxima instrucción
- IR (registro de instrucción): contiene la instrucción ejecutada
- PSW (estado): contiene información del estado del programa

El shell

El sistema operativo es el que recibe y lleva a cabo las llamadas al sistema. Los editores, compiladores, ensambladores, enlazadores e intérpretes de comandos definitivamente no son parte del sistema operativo, aunque son importantes y útiles. El shell es un intérprete de comandos, es la interfaz principal entre un usuario sentado en su terminal y el sistema operativo, a menos que el usuario utilice una interfaz gráfica (GUI). Existen muchos shells, incluidos sh, csh, ksh y bash. Todos ellos admiten la funcionalidad que se describe a continuación, que se deriva del shell original (sh).

Cuando un usuario inicia sesión, se inicia un shell. El shell presenta una terminal como entrada y salida estándar. Comienza escribiendo el indicador, el carácter con el signo pesos, que le dice al usuario que el shell está esperando para aceptar un comando. Si el usuario ahora escribe el

comando “*date*”, por ejemplo, el shell crea un proceso hijo y ejecuta el programa fecha como hijo. Mientras se ejecuta el proceso hijo, el shell espera a que termine. Cuando el proceso hijo termina, el shell vuelve a escribir el indicador e intenta leer la siguiente línea de entrada.

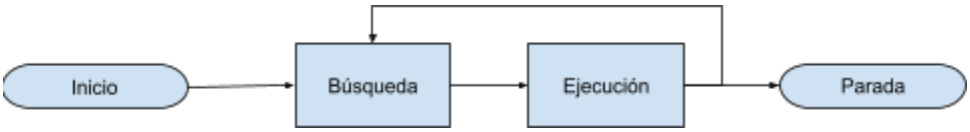
La mayoría de los sistemas operativos modernos, de propósito general, usan una GUI. De hecho, la GUI es solo un programa que se ejecuta en la parte superior del sistema operativo, como un shell.

Ciclo de ejecución de instrucciones

Al principio de cada ciclo, el procesador lee una instrucción de la memoria. El PC almacena la dirección de la siguiente instrucción a leer. El PC siempre se incrementa en 1 para leer la próxima instrucción, salvo que se indique lo contrario.

La instrucción leída se carga en el IR. Esta instrucción indica al procesador la acción a realizar. Éstas acciones pueden ser de tipo:

- **Procesador - memoria**: involucra transferencia de datos desde y hacia la memoria
- **Procesador - E/S**: involucra transferencia de datos desde y hacia un módulo de E/S
- **Procesamiento de datos**: operaciones aritméticas o lógicas
- **Control**: permiten alterar la secuencia de ejecución.



System Calls

Hemos visto que los sistemas operativos tienen dos funciones principales: proporcionar abstracciones a los programas de usuario y administrar los recursos de la computadora. En su mayoría, la primer función trata de proporcionar interfaces que permitan la interacción entre los programas de usuario y el sistema operativo; por ejemplo, crear, escribir, leer y eliminar archivos.

La segunda función, la gestión de recursos es en gran medida transparente para los usuarios y se realiza de forma automática.

La interfaz entre los programas de usuario y el sistema operativo está implementada principalmente como llamadas al sistema o **system calls.** Cada sistema operativo dispone de un conjunto de system calls que varían de uno a otro (aunque los conceptos subyacentes tienden a ser similares).

Es importante tener en cuenta que cualquier computadora con un único CPU puede ejecutar solo una instrucción a la vez. Si un proceso está ejecutando un programa de usuario en modo de usuario y necesita un servicio del sistema, como leer datos de un archivo, tiene que ejecutar una instrucción para transferir el control al sistema operativo. A continuación, el sistema operativo lleva a cabo la llamada al sistema y devuelve el control a la instrucción que sigue a la llamada al sistema.

Interrupciones

En el sentido más amplio, una interrupción es un mecanismo mediante el cual se puede interrumpir la ejecución normal del procesador. Cómo vimos al describir la multiprogramación, las interrupciones permiten realizar un uso más eficiente del procesador.

Hay 4 tipos de interrupciones:

- **De programa**: se producen como resultado de una ejecución de una instrucción. Por ejemplo, desbordamiento aritmético, división por cero, referenciar memoria fuera del espacio del programa, intento de apertura/lectura de un archivo, etc.
- **De E/S**: generada por un controlador de E/S para indicar la finalización de una operación.
- **Por temporizador**: permite al sistema operativo realizar ciertas funciones de forma regular.
- **Por fallo de hardware**

Ciclo de instrucción con interrupciones

Debido a la implementación de interrupciones, el procesador puede dedicarse a ejecutar otras instrucciones mientras que una operación de E/S se lleva a cabo.

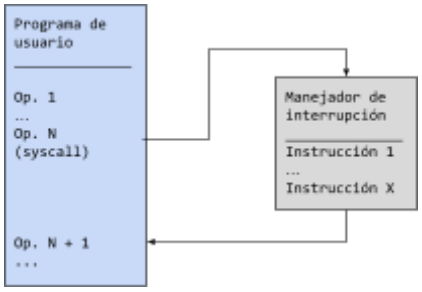
Para explicar este concepto, imagine un programa que debe, luego de procesar algunos datos, imprimirlos a través de una impresora o guardarlos en el disco rígido. Para esto el programa realizará una llamada al sistema para imprimir los datos.

En ese momento el sistema operativo toma el control y ejecuta una rutina de preparación del dispositivo de E/S donde se almacenan los datos que el programa quiere escribir en el buffer del dispositivo. Una vez terminada esta rutina, es decir, que el dispositivo tiene los datos, se comienza con la operación de escritura, momento en el cual el sistema operativo devuelve el control al programa de usuario que realizó la llamada al sistema para que continúe su ejecución.

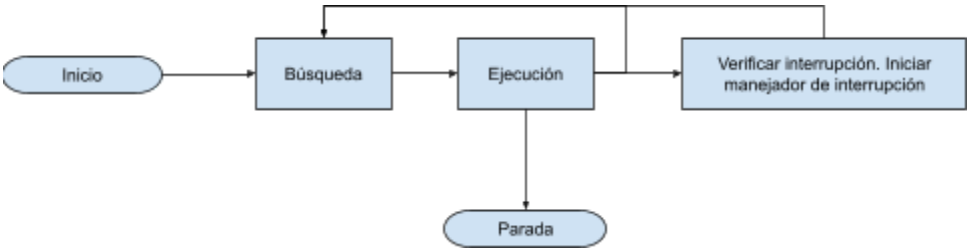
Cuando la operación de E/S termina, es decir que el dispositivo terminó de escribir los datos, el controlador del mismo manda una señal de *petición de interrupción* al procesador para indicar que está listo para realizar otra operación. El procesador termina de ejecutar la instrucción en

curso e interrumpe nuevamente el programa de usuario en ejecución para procesar la interrupción. Tener en cuenta que antes de comenzar a procesar la interrupción, el procesador debe salvar el estado del proceso actual para luego poder reanudar la ejecución.

Gracias a que las rutinas para manejo de interrupciones son parte del sistema operativo, los programas de usuario no deben preocuparse por implementar nada de esto. La ejecución de un programa de usuario se vería típicamente como:



Si al ciclo de instrucción explicado en el punto anterior le sumamos las interrupciones, se ve modificado de la siguiente manera:



Técnicas de comunicación de E/S[2]

Existen 3 técnicas principales para el manejo de la E/S de datos:

1. E/S programada
2. E/S por interrupciones
3. DMA o acceso directo a memoria

E/S programada

Con esta técnica, el procesador es responsable de extraer los datos de la memoria principal en una operación de salida y de almacenarlos en ella en una operación de entrada. El mayor problema o desventaja es que el proceso permanece en un bucle de comprobación constante hasta que haya finalizado la operación (**busy waiting**). Podríamos decir que en cierta forma el

[2] Entrada / Salida

procesador se acopla al dispositivo de entrada-salida y se lo mantiene ocupado innecesariamente.

E/S por interrupciones

Al procesador ser interrumpido por el módulo de E/S, representa una mejora respecto de la E/S programada ya que el procesador no está todo el tiempo consultando si la operación finalizó.

Desde el punto de vista de un módulo de E/S, se recibe la orden de lectura (o escritura) del procesador. Se comienza a leer los datos de un periférico asociado. Una vez que los datos están en el registro de datos del módulo, genera una interrupción al procesador. A partir de ese momento el módulo queda a la espera de que el procesador solicite los datos. El procesador pide los datos al módulo de E/S quien los coloca en el bus de datos.

Desde el punto de vista del procesador, el procesador genera la orden de lectura de datos, salva el contexto de ejecución del programa actual (PC, PSW, etc) y pasa a realizar otra acción. Al finalizar cada instrucción comprueba si hay alguna interrupción, en caso de que la haya, salva el contexto del programa actual y comienza a ejecutar el programa de manejo de interrupciones. El procesador lee la palabra del módulo de E/S y la almacena en memoria, restaura el contexto del programa de usuario que había realizado la petición de datos y reanuda la ejecución.

Si bien esta técnica es más eficiente que la E/S programada, todavía consume tiempo valioso del procesador ya que para que el programa pueda obtener los datos sí o sí interviene el procesador.

DMA

DMA o acceso directo a memoria (por sus siglas en inglés) es una técnica conveniente cuando se requiere transferir grandes volúmenes de datos. Esta técnica funciona de la siguiente manera. Cuando el procesador requiere leer o escribir datos, genera una orden para el módulo de DMA enviando la siguiente información:

1. Tipo de operación (escritura o lectura)
2. Dispositivo de E/S involucrado
3. Posición inicial en donde leer o escribir los datos
4. Cantidad de palabras a leer o escribir.

A continuación el procesador continúa con la ejecución de otro programa. El módulo de DMA, se encargará de transferir los datos desde o hacia la memoria sin utilizar el procesador, por lo que el procesador sólo se involucra al principio y al final de toda la operación.

Los 5 pilares de un sistema operativo

Los 5 conceptos en los que se basa el desarrollo de un sistema operativo son:

1. Procesos
2. Gestión de almacenamiento
3. Protección y seguridad de la información
4. Planificación y gestión de los recursos
5. Estructura del sistema

Procesos

Definición	**¿Qué es un proceso?** Es una unidad de actividad caracterizada por un solo hilo secuencial de ejecución, un estado actual y un conjunto de recursos del sistema asociados.

En otras palabras, un proceso es un programa en ejecución, que tiene asociado un espacio de direcciones de memoria (*address space*) que puede leer y escribir. Este espacio de direcciones contiene las instrucciones, los datos y la pila de ejecución (stack) del proceso. También existen algunos registros asociados como el PC (program counter), SP (stack pointer) y varios más, necesarios para posibilitar la ejecución del proceso.

Cada proceso está formado por 3 componentes:

- Instrucciones
- Datos
- Contexto de ejecución.

El contexto de ejecución o estado del proceso, es el conjunto de datos que utiliza el sistema operativo (junto al procesador) para gestionar y ejecutar el proceso. Son los registros del procesador como PC, IR, PSW. Esta información está separada del proceso ya que contiene información que el proceso no debe poder manipular, como por ejemplo el modo de ejecución, que es un bit que se encuentra dentro del PSW.

Por otra parte, el procesador mantiene, además de los registros mencionados, otros registros que hacen al contexto del programa como ser el registro que guarda la posición inicial de memoria de un proceso (base) y el registro que guarda la posición final de memoria del proceso (límite).

El PC (program counter) y todas las referencias a datos de un proceso se interpretan de forma relativa al registro base; y es gracias a estos 2 registros (base y límite) que el procesador puede realizar las tareas de verificación respecto de que un proceso no lea memoria por fuera de su espacio (*address space*).

Para tener un acercamiento a cómo funcionan los procesos, el mejor ejemplo es pensar en un sistema multiprogramado. Periódicamente, el sistema operativo decide detener la ejecución de un proceso y comenzar a ejecutar otro, por ejemplo, porque el primero ha tenido más tiempo de CPU de lo que le correspondía en el último segundo. Cuando un proceso se suspende temporalmente de esta manera, debe luego reiniciarse exactamente en el mismo estado que tenía cuando se detuvo. Esto significa que toda la información sobre el proceso debe guardarse explícitamente en algún lugar durante la suspensión. Por ejemplo, el proceso puede tener varios archivos abiertos para leer a la vez. Asociado con cada uno de estos archivos hay un puntero que da la posición actual (es decir, el número del byte o registro que se leerá a continuación). Cuando un proceso se suspende temporalmente, todos estos punteros deben guardarse para que una llamada de lectura ejecutada después de reiniciar el proceso lea los datos adecuados. En muchos sistemas operativos, toda la información sobre cada proceso, que no sea el contenido de su propio espacio de direcciones, se almacena en una tabla del sistema operativo llamada tabla de procesos (**process table**), que es una matriz de estructuras, una para cada proceso.

Gestión de almacenamiento

Un sistema operativo posee 5 responsabilidades esenciales respecto del almacenamiento:

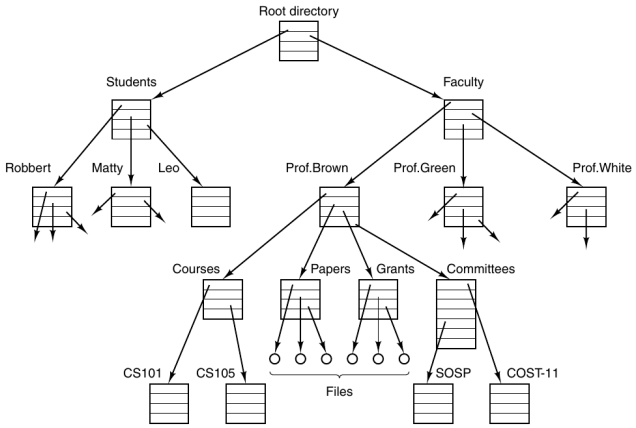
- ❏ Aislamiento de procesos: evitar que los procesos interfieran en el espacio de memoria de otro proceso
- ❏ Asignación y gestión automática: la asignación de memoria debe ser transparente al programador
- ❏ Programación modular: el sistema operativo debe dar soporte a la programación modular.
- ❏ Protección y control de acceso: en ciertos escenarios es deseable que un proceso pueda direccionar la memoria de otro. El sistema operativo debe permitir que varios usuario accedan de distintas formas a porciones de la memoria
- ❏ Almacenamiento a largo plazo: este aspecto lo lleva a cabo a través del uso de archivos y tiene que ver con almacenar información incluso cuando el equipo se haya apagado.

Archivos

El almacenamiento a largo plazo, el sistema operativo lo lleva a cabo a través de la implementación de archivos (ficheros).

Definición	**¿Qué es un archivo?** Es un concepto lógico, conveniente para el programador y es una unidad útil para el control de acceso y protección para el sistema operativo

La mayoría (o todos) los sistemas operativos modernos implementan además, el concepto de directorio como una forma de agrupar archivos. Las entradas de cada directorio pueden ser archivos u otros directorios posibilitando de esta manera, el armado de una estructura jerárquica de archivos y directorios.



La jerarquía mencionada permite establecer permisos de lectura/escritura tanto sobre los archivos como los directorios de forma tal que se pueda permitir o restringir el acceso y las acciones que se realizan sobre ellos.

Un aspecto a destacar, es que cada proceso tiene un directorio de trabajo actual (working directory), a partir del cual se buscan todas las referencias a archivos cuya ruta no comienzan con una barra. Por ejemplo, si /Faculty/Prof.Brown fuera el directorio de trabajo, entonces el uso del nombre “Courses/CSlOl” haría referencia al mismo archivo que si indicáramos el nombre completo de la ruta (PATH absoluto), por ejemplo “/Faculty/Prof.Brown/Courses/CSlOl”.

Cuando una ruta comienza con “/” significa que se debe buscar desde el directorio raíz.

Los procesos pueden cambiar su directorio de trabajo utilizando una llamada al sistema y especificando el nuevo directorio de trabajo. Antes de poder leer o escribir un archivo, el mismo debe abrirse, momento en el que se verifican los permisos. Si se permite el acceso, el sistema devuelve un entero llamado descriptor de archivo para usar en operaciones posteriores. Si el acceso está prohibido, se devuelve un código de error.

Memoria Virtual

Otro aspecto interesante y útil sobre la gestión de almacenamiento que realiza un sistema operativo es el de la ***memoria virtual.*** La memoria virtual, es un concepto que permite a los programas direccionar la memoria desde un punto de vista lógico, es decir, sin importar la cantidad de memoria principal disponible.

La memoria virtual fue concebida como un método para poder mantener múltiples trabajos en memoria principal al mismo tiempo (*multiprogramación*), para acortar los tiempos de espera entre la ejecución de un proceso y otro.

Con el surgimiento de la multiprogramación, los diseñadores de sistemas operativos se vieron forzados a pensar un esquema de asignación de memoria distinto y más eficiente que el usado hasta ese momento con los sistemas por lotes. Hasta ahora sabemos que cuando el sistema operativo necesita cargar un programa y no tiene memoria disponible, lo que hace es sacar un proceso de la memoria y guardarlo en disco. Bueno, este método no es realmente eficiente, por lo que se pensó un esquema donde la memoria esté fraccionada en bloques de tamaño fijo llamados "***página***". De esta forma se daba origen a los sistemas de paginación donde un programa que es cargado en memoria puede ocupar varias páginas y además, cuando se requiere espacio no es necesario reemplazar todo un programa sino que se puede reemplazar sólo algunas páginas del mismo. Este esquema también modificó la forma de direccionar las palabras ya que a partir de ahora se direccionan a través de un número de página más un desplazamiento dentro la página.

¿Qué pasa cuando un programa necesita una instrucción o un dato que se encuentra en una página que fue llevada a disco?

Esto es lo que se conoce como ***page fault*** o fallo de página.

Por cada instrucción o dato requerido por el programa en ejecución, el sistema operativo lo busca primeramente dentro de las páginas del programa cargadas en memoria principal. Si no lo encuentra (recordemos que una dirección de memoria se compone de número de página + desplazamiento dentro de la misma) se dispara un mecanismo para recuperar una o varias páginas de la memoria secundaria que contengan la dirección buscada. Esta página deberá reemplazar a otra página.

Protección y seguridad de la información

El trabajo de un sistema operativo de cara a la seguridad y protección, puede agruparse en:

- Disponibilidad: relacionado con la protección del sistema frente a interrupciones

- Confidencialidad: proteger el acceso de los datos de aquellos que no tienen autorización de acceso.
- Integridad: proteger la modificación de los datos de aquellos que no tienen autorización.
- Autenticidad: verificar la identidad de los usuarios

Planificación y gestión de los recursos

El sistema operativo es responsable por la correcta administración de los recursos (memoria, dispositivos de E/S, procesador) para su uso por los distintos procesos activos. Un sistema operativo debe garantizar:

- Equitatividad: el sistema operativo debe garantizar que los procesos tengan un acceso equitativo a los recursos.
- Respuesta diferencial: el sistema operativo debe poder discriminar los procesos y otorgar distintas prioridades a cada uno.
- Eficiencia: el sistema operativo debe maximizar la productividad, minimizar los tiempos de respuesta.

Para poder cumplir con estos requisitos, el sistema operativo debe poder planificar de forma correcta los procesos que desean ejecutarse. Para esto cuenta con colas de prioridad, que son listas de procesos que esperan usar el procesador. Estas colas pueden ser a largo plazo, donde se almacenan los procesos que ingresan y que aún no están listos para ejecutar, así como también aquellos que quedan esperando por algún dispositivo de E/S. La otra lista es a corto plazo donde se alojan los procesos que están listos para ejecutar. Es responsabilidad del sistema operativo seleccionar uno de los procesos, de la lista a corto plazo, para ser ejecutado.

Estructura del sistema

El tamaño de un sistema operativo y las dificultades que afronta llevaron a tener en cuenta las siguientes cuestiones para su desarrollo:

- Modularidad para organizar el desarrollo, facilidad de mantenimiento, corrección de errores.
- Interfaces bien definidas y sencillas: facilita la programación. Permite modificar un módulo con mínimo impacto en los otros módulos
- Estructura jerárquica (niveles): permite una mejor abstracción de la información, separar sus funciones.

Esta última característica conduce a que cada nivel realice un subconjunto de funciones del sistema operativo y cada nivel delega en el nivel inmediatamente inferior la ejecución de

funciones más primitivas. Dicho en otras palabras, cada nivel proporciona servicios a la capa inmediatamente superior.

La forma en que cada sistema operativo implementa estos conceptos varía enormemente. Una propuesta teórica de dichos niveles es la siguiente:

Nivel	Nombre	Objetos	Parte de
13	Intérprete de comandos	programas de usuario, entornos de programación	Sistema operativo
12	Procesos de usuario		
11	Directorios	Directorios	
10	Dispositivos	Periféricos	
9	Sistema de archivos	Archivos	
8	Comunicaciones	Tuberías	
7	Memoria virtual	Segmentos y páginas	
6	Almacenamiento secundario (discos)	Bloques de datos	
5	Procesos primitivos	Semáforos, lista de procesos	
4	Interrupciones	Programa de gestión de interrupciones	Hardware
3	Procedimientos	Pila de llamadas, registros de activación	
2	Conjunto de instrucciones	Pila de evaluación, intérprete de microprogramas	
1	Circuitos electrónicos	Registros del procesador, buses, etc.	

Cabe aclarar que del nivel 1 al 4 constituyen el hardware del procesador y no forman parte del sistema operativo. Exceptuando el nivel 4 donde se empiezan a mezclar elementos del sistema operativo como el manejador de interrupciones.

Diseño de un sistema operativo

A continuación se describen cinco diseños diferentes que se han probado en la construcción de sistemas operativos. Los cinco diseños son sistemas monolíticos, sistemas en capas, máquinas virtuales, exokernels y sistemas cliente-servidor.

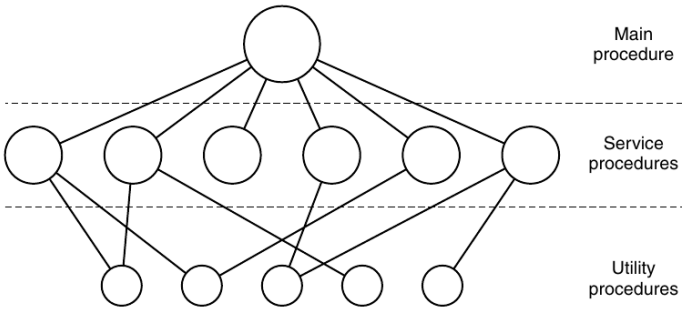
Monolíticos

El sistema operativo está escrito como una colección de procedimientos, cada uno de los cuales puede llamar a cualquier otro procedimiento cuando lo necesite.

Cuando se utiliza esta técnica, cada procedimiento en el sistema tiene una interfaz bien definida en términos de parámetros y resultados, y cada uno es libre de llamar a cualquier otro.

Para construir el sistema operativo cuando se usa este enfoque, primero se compilan todos los procedimientos individuales, o archivos que contienen los procedimientos, y luego se vinculan todos juntos en un solo archivo usando el enlazador del sistema.

En términos de ocultamiento de información, esencialmente no hay ninguno: todos los procedimientos son visibles para todos los demás procedimientos (a diferencia de una estructura que contiene módulos o paquetes, en los que gran parte de la información está oculta dentro de los módulos, y solo los puntos de entrada designados oficialmente pueden llamarse desde fuera del módulo). Sin embargo, incluso en sistemas monolíticos, es posible tener al menos una pequeña estructura. En este modelo, para cada llamada al sistema hay un procedimiento de servicio que se encarga de ello. Esta división de los procedimientos en tres capas se muestra en la siguiente figura:



En capas

Es una generalización del esquema monolítico, donde una capa se construye sobre la anterior. El primer sistema construido con este esquema fue llamado THE (Technische Hogeschool Eindhoven). Fue construído por E. W. Dijkstra (1968) y sus alumnos. El sistema tenía 6 capas

Capa	Función
5	El shell del usuario
4	Gestiona los programas de usuario. Proporciona bibliotecas para manejo de memoria, I/O, etc.
3	Gestión de la Entrada / Salida
2	Maneja la comunicación entre procesos
1	Gestión de memoria
0	Gestión de procesos y multiprogramación

Otro ejemplo que implementó este diseño fue el sistema MULTICS, que se organiza en forma de anillos concéntricos, donde los más internos tenían más privilegios que los más externos. Cuando un proceso de un anillo externo necesitaba invocar un proceso de un anillo interno, debía realizar el equivalente a una llamada al sistema.

Microkernels

Con el enfoque en capas, los diseñadores pueden elegir dónde trazar el límite entre el espacio del usuario y el del kernel. Tradicionalmente, todas las capas iban en el kernel, pero eso no es necesario. De hecho, resulta conveniente que la capa del kernel posea menos responsabilidades porque un error en el kernel puede derribar el sistema instantáneamente. Por el contrario, los procesos de usuario se pueden configurar para que tengan menos poder de acción, de modo que un error no sea fatal.

La idea básica detrás del diseño del microkernel es lograr una alta confiabilidad dividiendo el sistema operativo en módulos pequeños y bien definidos, solo uno de los cuales, el microkernel, se ejecuta en modo kernel y el resto se ejecuta como procesos de usuario. La justificación detrás de este enfoque es que al ejecutar cada controlador de dispositivo y sistema de archivos como un proceso de usuario, un error en uno de estos sólo bloquea ese componente, pero no todo el sistema.

Por el contrario, en un sistema monolítico con todos los controladores en el kernel, un controlador de audio defectuoso puede hacer referencia fácilmente a una dirección de memoria no válida y hacer que el sistema se detenga instantáneamente.

Este diseño de sistema operativo, es particularmente común en sistemas de tiempo real, industrial, aviónica y aplicaciones militares que son de misión crítica y tienen requisitos de

confiabilidad muy altos. Algunos de los microkernels más conocidos son Integrity, K42, L4, PikeOS, QNX, Symbian y MINIX 3.

Cliente-Servidor

Una variación de la idea del microkernel es distinguir dos clases de procesos, los de tipo servidor, que proporciona algún servicio, y los de tipo cliente, que utilizan estos servicios. Este modelo se conoce como modelo cliente-servidor. A menudo, la capa más baja es un microkernel, pero no es necesario.

La comunicación entre clientes y servidores se realiza a menudo mediante el paso de mensajes. Para obtener un servicio, un proceso cliente construye un mensaje diciendo lo que quiere y lo envía al servicio apropiado. El servicio luego hace el trabajo y envía la respuesta.

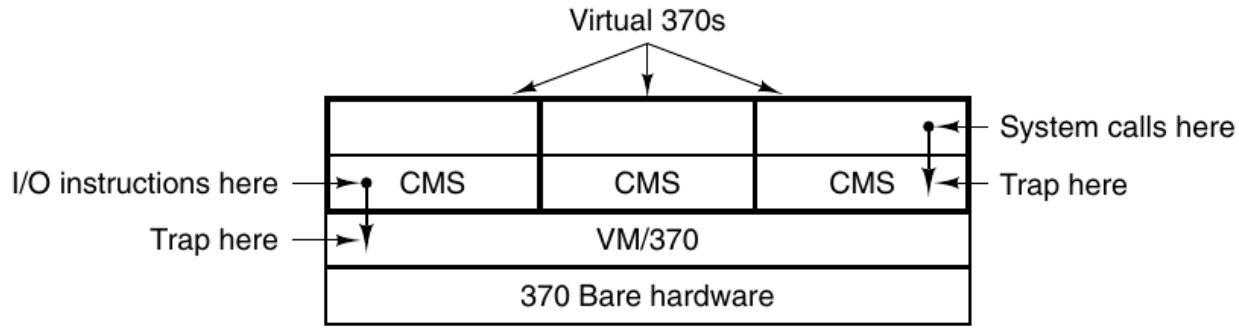
Máquina Virtual

El primer sistema en implementar este tipo de diseño surgió como respuesta a implementar un sistema de tiempo compartido. Originalmente llamado CP / CMS y luego rebautizado como VM / 370, este sistema se basó en la siguiente observación: un sistema de tiempo compartido proporciona:

1. multiprogramación
2. una máquina extendida con una interfaz conveniente para el uso del hardware.

La esencia del VM / 370 era separar completamente estas dos funciones.

El corazón del sistema, conocido como monitor de la máquina virtual, se ejecuta en el hardware básico y realiza la multiprogramación, proporcionando no una, sino varias máquinas virtuales a la siguiente capa, como se muestra en la siguiente figura. Sin embargo, a diferencia de todos los demás sistemas operativos, estas máquinas virtuales no son máquinas extendidas, con archivos y otras características interesantes. En cambio, son copias exactas del hardware básico, incluido el modo kernel / usuario, E / S, interrupciones y todo lo demás que tiene la máquina real.



Debido a que cada máquina virtual es idéntica al hardware real, cada una puede ejecutar cualquier sistema operativo que se ejecute directamente en el hardware básico. Diferentes máquinas virtuales pueden ejecutar, y con frecuencia lo hacen, diferentes sistemas operativos.

Cuando un programa ejecuta una llamada al sistema, la llamada queda atrapada en el sistema operativo en su propia máquina virtual, no en VM / 370, tal como lo haría si se estuviera ejecutando en una máquina real en lugar de una virtual. A continuación, se emiten las instrucciones de E / S de hardware normales para leer su disco virtual o lo que sea necesario para realizar la llamada. Estas instrucciones de E / S son capturadas por VM / 370, que luego las ejecuta como parte de su simulación del hardware real.

Al hacer una separación completa de las funciones de multiprogramación y proporcionar una máquina extendida, cada una de las piezas puede ser mucho más simple, más flexible y más fácil de mantener.

Exokernel

En lugar de clonar la máquina real, como se hace con las máquinas virtuales, otra estrategia es particionarla, en otras palabras, darle a cada usuario un subconjunto de los recursos.

En la capa inferior, que se ejecuta en modo kernel, hay un programa llamado exokernel, su trabajo es asignar recursos a las máquinas virtuales y luego verificar los intentos de usarlos para asegurarse de que ninguna máquina esté tratando de usar los recursos de otra persona. Cada máquina virtual a nivel de usuario puede ejecutar su propio sistema operativo, como en VM / 370 y Pentium virtual 8086s, excepto que cada una está restringida a usar solo los recursos que ha solicitado y asignado.

La ventaja del esquema de exokernel es que ahorra una capa de mapeo. En los otros diseños, cada máquina virtual piensa que tiene su propio disco, con bloques que van desde 0 hasta un máximo, por lo que el monitor de la máquina virtual debe mantener tablas para reasignar direcciones de disco (y todos los demás recursos). Con el exokernel, esta reasignación no es necesaria. El exokernel sólo necesita realizar un seguimiento de qué máquina virtual se ha asignado a qué recurso. Este método todavía tiene la ventaja de separar la multiprogramación (en el exokernel) del código del sistema operativo del usuario (en el espacio de usuario), pero con menos sobrecarga, ya que todo lo que tiene que hacer el exokernel es mantener las máquinas virtuales fuera del alcance de los demás..

Unidad II

~ Procesos ~

"Si no puedes describir lo que está haciendo como un proceso, no sabes lo que está haciendo".

~ W. Edwards Deming

Procesos

Introducción

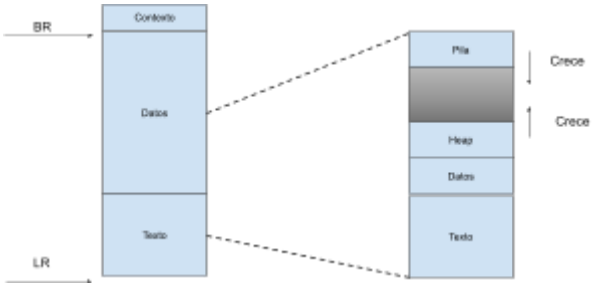
Ahora estamos a punto de embarcarnos en un estudio detallado de cómo se diseñan y construyen los sistemas operativos. El concepto más importante y central en cualquier sistema operativo es el de **proceso**: una abstracción de un programa en ejecución. Todo lo demás depende de este concepto.

En la actualidad, las computadoras pueden hacer varias cosas al mismo tiempo. Mientras se ejecuta un programa de usuario, una computadora también puede leer desde un disco y enviar texto a una pantalla o impresora. En un sistema multiprogramado, el CPU también cambia de un programa a otro, ejecutando cada uno durante decenas o cientos de milisegundos. Si bien, estrictamente hablando, en un instante de tiempo, el CPU está ejecutando sólo un programa, en el transcurso de 1 segundo, puede ejecutar varios programas, dando así a los usuarios la ilusión de paralelismo. A veces, se habla de pseudoparalelismo en este contexto, para contrastarlo con el verdadero paralelismo de hardware de los sistemas multiprocesador (que tienen dos o más CPU que comparten la misma memoria física).

Definición

De manera informal, podemos decir que un proceso es un programa en ejecución. Sin embargo, un proceso es mucho más que eso. Un proceso no sólo está compuesto por el código del programa sino que además posee datos globales y locales, información de contexto, áreas de memoria dinámica, en conclusión no es sólo código.

Una representación de la imagen de un proceso en memoria podría ser la siguiente:



La sección llamada “Contexto”, contiene todos los registros utilizados por el sistema operativo para poder gestionar el proceso, tales como el PC (program counter), IR (instruction record), etc.

La sección “Texto” contiene el código del programa, o mejor dicho las instrucciones que representan dicho programa.

La sección “Datos” es utilizada para almacenar datos y variables globales de proceso.

Por último tenemos las secciones “Pila” (stack) y “Heap” (o montículo, pero vamos a llamarlo heap) son utilizadas para almacenar datos temporales. La “Pila” es utilizada para almacenar variables globales, llamadas a funciones, parámetros, puntos de retorno, etc. y el “Heap” es utilizado para la asignación dinámica de memoria (estructuras dinámicas como pilas, colas y listas).

Corolario	**Programa vs Proceso** Un programa en sí mismo no es un proceso. Un programa es una entidad pasiva, un archivo que contiene instrucciones que se encuentran almacenadas en disco. Un proceso por su parte, es una entidad activa, que posee un contexto y recursos asociados.

Descripción y control de procesos

En la Unidad I vimos algunas definiciones para el concepto de proceso. También podríamos decir que un proceso es una entidad compuesta por un conjunto de elementos, principalmente el código del programa, sus datos y el ***bloque de control de proceso (BCP)***. Este último elemento es una estructura de datos que el sistema operativo crea y utiliza para poder gestionar el proceso. Los datos más importantes que incluye el BCP son:

- **Identificador (PID)**: identificador único del proceso.
- **Estado**: el estado en el que se encuentra el proceso
- **Prioridad**: nivel de prioridad relativo al resto de los procesos
- **Contador de programa (PC)**: dirección de memoria de la siguiente instrucción del programa a ejecutar
- **Datos de contexto**: datos que están presentes en los registros del procesador cuando el proceso está corriendo.

- **Información de estado de E/S**: incluye peticiones de E/S pendientes, archivos en uso, etc.

Otros datos que contiene:

- **Punteros a memoria**: incluye punteros al código del programa y datos, y bloques de memoria compartida.
- **Datos de auditoría**: tiempo de procesador, tiempo que está en ejecución, etc.

El punto más importante es que a través del **BCP** el sistema operativo puede gestionar cada uno de los procesos y proporcionar multiprogramación. Por ejemplo, cuando un proceso es suspendido para dar paso a otro proceso, los datos actuales de los registros del procesador se guardan en *los datos de contexto* del proceso que se va a suspender y del proceso elegido para ejecutar, se recuperan los datos del contexto que son cargados en los registros del procesador para continuar con su ejecución.

Estados de un proceso

Una de las responsabilidades del sistema operativo, si no es la más importante, es controlar la ejecución de los procesos y asignar recursos a los mismos. Para esto, el primer paso es definir qué comportamiento se desea que tengan.

Modelo de 2 estados

El modelo más simple que podemos construir es aquel en el que un proceso está siendo ejecutado por el procesador o no, es decir, en este modelo un proceso sólo puede estar en 2 estados *Ejecutando* o *No Ejecutando*.

Cuando el sistema operativo crea un proceso, crea su BCP y lo inserta al sistema en estado No Ejecutando. A partir de ese momento, el proceso existe y está esperando su oportunidad para ejecutar. En algún momento, el proceso que se está ejecutando será interrumpido y el *dispatcher* seleccionará otro proceso para ejecutar. El proceso interrumpido pasará a estado No Ejecutando.

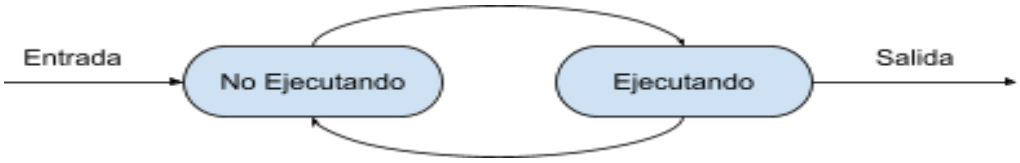
Definición	**¿Qué es el dispatcher?** Es un pequeño programa que se encarga de intercambiar los procesos en ejecución

A pesar de la simpleza de este modelo, vemos que para poder realizar el intercambio de procesos, el sistema operativo necesita tener información de los mismos que le permita

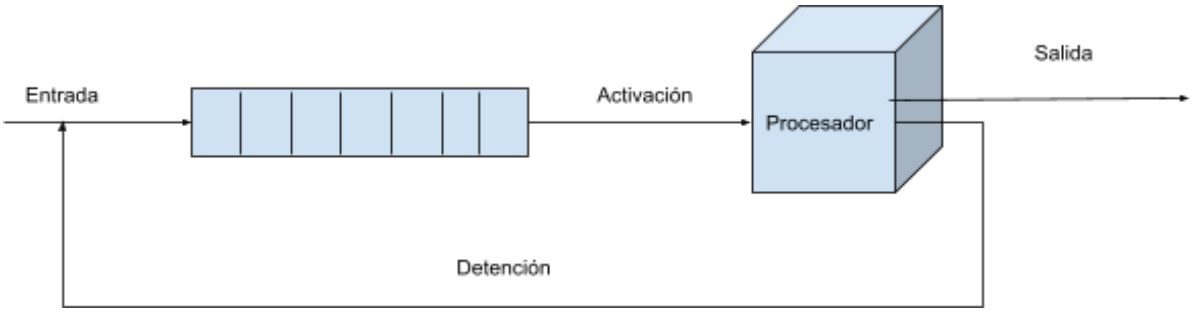
"seguirlos", restaurarlos, etc. Es por esto, entre otras cosas, que surge el BCP que incluye el estado actual y su localización en memoria.

Los procesos que no se están ejecutando, esperan en una cola su turno para ejecutarse. Los elementos de estas colas, no son los procesos propiamente dicho sino que son punteros a los BCP de los procesos.

El siguiente gráfico es un esquema representativo del proceso descrito.



Una representación de la cola utilizada por el sistema operativo para los procesos detenidos:



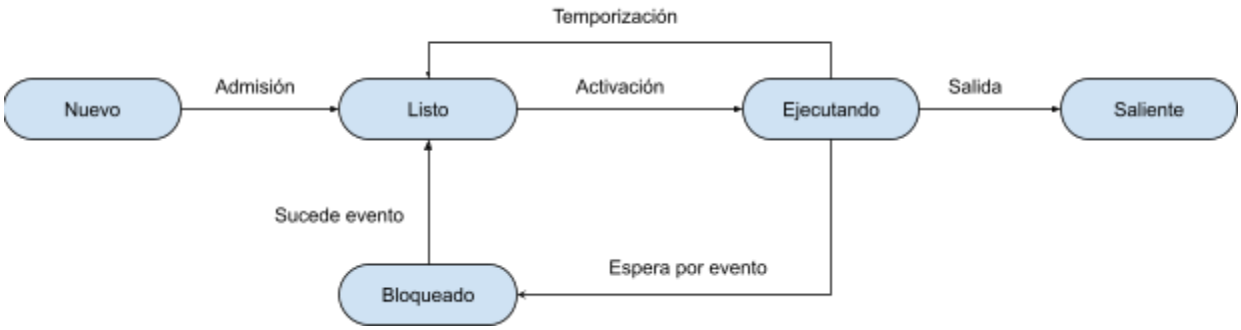
La cola en la que esperan los procesos, es una lista de tipo FIFO (primero que entra, primero que sale) y el procesador opera siguiendo una estrategia cíclica (round-robin) sobre todos los procesos.

Modelo de 5 estados

El modelo presentado anteriormente no resulta muy efectivo para gestionar los procesos ya que algunos de los procesos que se encuentran en la cola de espera, en estado *No Ejecutando*, podrían estar listos para ejecutar mientras que otros pueden estar en dicho estado debido a que se encuentran bloqueados esperando por una operación de E/S.

Para sortear este inconveniente, la solución es desdoblar el estado de *No Ejecutando* en 2 estados nuevos: *Listo* y *Bloqueado*. Para poder gestionar los procesos correctamente se agregan 2 estados más que son *Nuevo* y *Saliente*.

- **Nuevo**: un proceso que se acaba de crear y que aún no ha sido admitido en el grupo de procesos ejecutables por el sistema operativo. Se trata de un nuevo proceso que no ha sido cargado en memoria principal, aunque su BCP si ha sido cargado.
- **Listo**: un proceso que se encuentra listo para ejecutar.
- **Ejecutando**: un proceso que está actualmente en ejecución.
- **Bloqueado**: un proceso que no puede ejecutar hasta que se cumpla un evento determinado o se complete una operación de E/S.
- **Saliente**: un proceso que fue detenido o ha sido abortado.



Los estados *Nuevo* y *Saliente* son útiles para gestionar los procesos y poder administrar mejor los recursos del sistema.

El estado Nuevo

El estado Nuevo se corresponde con un proceso que acaba de ser definido, por ejemplo en un sistema de tiempo compartido, si un nuevo usuario intenta ingresar en el sistema, o en un sistema operativo por lotes si se intenta crear un nuevo trabajo.

¿Qué ocurre internamente?

Cuando el sistema operativo recibe la solicitud de un nuevo proceso, le asocia un nuevo identificador (PID) al proceso, se reservan y construyen todas las tablas utilizadas para gestionar dicho proceso.

En este punto, el proceso se encuentra en estado nuevo. Esto significa que se han construído todas las estructuras necesarias para la gestión del proceso, pero el proceso en sí no ha sido cargado en memoria principal, es decir, su código no ha sido cargado ni se ha reservado espacio en memoria principal para los datos del proceso. Cuando un proceso se encuentra en estado “Nuevo”, el programa permanece en almacenamiento secundario.

El sistema operativo puede mantener un proceso en estado “Nuevo” por razones de rendimiento o falta de memoria principal u otros recursos necesarios para la ejecución del proceso.

Analicemos algunas transiciones no triviales:

Ejecutando → Listo: la razón más común es que el proceso haya alcanzado su tiempo máximo ininterrumpido de ejecución (utilización del procesador). Otras razones pueden ser cambios en los niveles de prioridad de los procesos

Listo (o Bloqueado) → Saliente: imagine un escenario donde existe un proceso B que es hijo de un proceso A, y el proceso B se encuentra en estado Listo. Si el proceso A termina o ejecuta alguna instrucción que haga finalizar al proceso B, el mismo pasará de estado Listo a Saliente.

Tanto el estado Listo como el estado Bloqueado, manejan colas (FIFO) de espera para los procesos. De hecho, el sistema operativo crea colas por cada evento por el cual esperan los procesos. Cuando ocurre algún evento X todos los procesos en estado Bloqueado que esperaban por dicho evento son movidos al estado *Listo*.

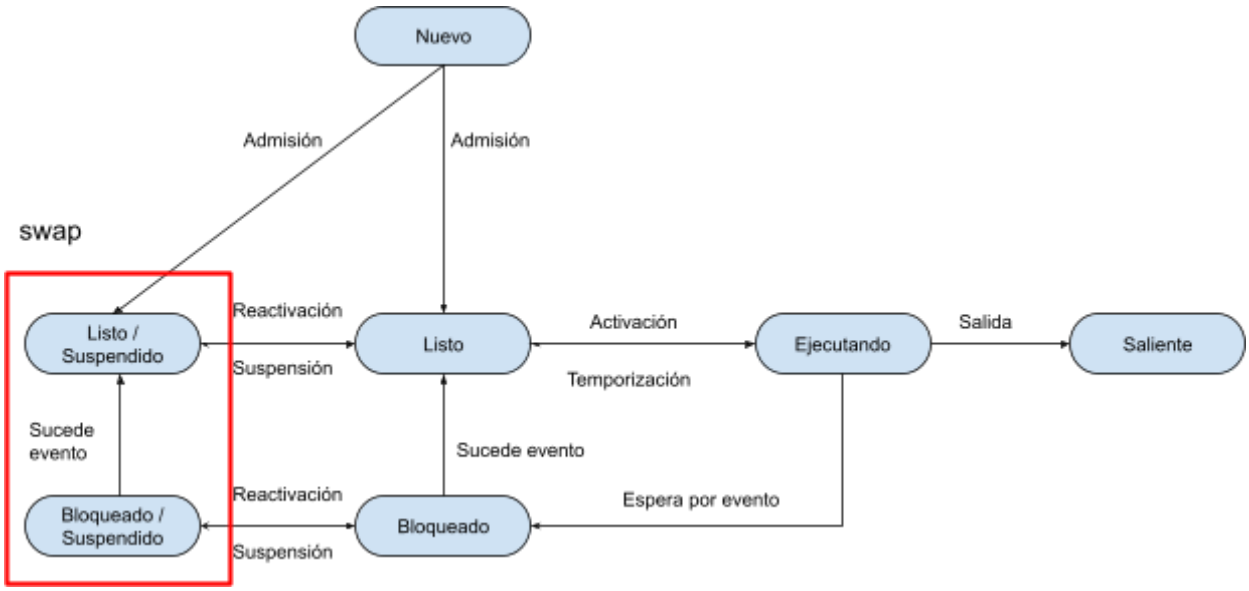
Modelo de 7 estados

Para entender el siguiente modelo, imagine por un momento que no existe la memoria virtual y por ende los procesos deben estar cargados completamente en memoria principal. Si en un momento determinado la memoria principal estuviese cargada por completo y todos los procesos se encontraran en estado *Bloqueado*, el procesador estaría ocioso y no se podrían admitir nuevos procesos.

Para resolver este problema surge el *swapping* (o memoria de intercambio) que implica mover una parte o todo el proceso de memoria principal a disco. Cuando ninguno de los procesos que se encuentra en memoria principal, se encuentra en estado *Listo*, el sistema operativo mueve uno de los procesos en estado *Bloqueado* a un nuevo estado que será el de *Suspendido* (disco). De esta manera se obtiene espacio para admitir un nuevo proceso en el sistema.

Tener en cuenta que el swapping es una operación de E/S.

Con este diseño se agregan nuevos estados y nuevas transiciones que reflejamos en el gráfico siguiente.



Algunas transiciones a destacar son:

Listo/Suspendido → Listo: cuando no hay procesos en estado listo el sistema tomará un proceso del estado Listo/Suspendido. Otro motivo puede ser que un proceso Listo/Suspendido tenga más prioridad que un proceso Listo

Nuevo → Listo/Suspendido: puede ocurrir que no haya espacio suficiente en memoria para el nuevo proceso y por eso el sistema operativo decide suspenderlo.

Bloqueado/Suspendido → Bloqueado: en ocasiones el sistema operativo puede traer a memoria principal un proceso suspendido basándose en estadísticas. Imagine que un proceso termina, liberando espacio en memoria y un proceso suspendido posee mayor prioridad que cualquiera de los procesos en estado Listo. Basándose en estadísticas el sistema operativo puede estimar que el evento que espera el proceso suspendido va a ocurrir y por esto es que un proceso suspendido en disco es traído a memoria principal.

Estructuras de control de procesos

Como mencionamos anteriormente, mucha de la información de un proceso, es utilizada por el sistema operativo para su gestión. Esta información, agrupada en el BCP, es cargada en los registros del procesador cuando el proceso está ejecutándose y vuelta a almacenar en el BCP cuando el proceso deja de ejecutarse (vuelve a estado Listo o es Bloqueado o Suspendido)

Ahora bien, además de estos atributos que se almacenan en el BCP ¿Qué información incluye la representación de un proceso?

Como mínimo, un proceso incluye un programa y posiciones de memoria para las variables locales, constantes y variables globales. Adicionalmente, un proceso incluye una pila que se utiliza para registrar las llamadas a funciones. En esta pila se almacenan las direcciones de memoria de retorno y los parámetros utilizados en cada llamada.

Podemos agrupar la información del bloque de control del proceso en 3 categorías:

- Identificación del proceso
- Información de estado del proceso: indica los contenidos de los registros del procesador cuando el proceso se está ejecutando. Algunos de estos registros son registros de control y estado, punteros de pila.
- Información de control del proceso: esta información es utilizada por el sistema operativo para controlar y coordinar los procesos activos.

Control de procesos

Modos de ejecución

El procesador posee 2 modos de ejecución, **el modo núcleo** asociado con el sistema operativo y **el modo usuario** asociado con los procesos de usuario. En modo núcleo, el software tiene el control absoluto del procesador (instrucciones, registros).

¿Cómo conoce el procesador en qué modo está ejecutando?

Dentro de la información de estado del proceso, se encuentra el PSW (program status word o palabra de estado del programa). El PSW contiene un bit que indica el modo de ejecución.

¿Cómo puede modificarse el modo de ejecución?

Este bit cambia en respuesta a determinados eventos como por ejemplo una llamada al sistema por parte del proceso que se está ejecutando.

Cambio de contexto (context switch)

A modo de introducción podemos decir que un cambio de contexto ocurre cada vez que el sistema operativo obtiene el control sobre el proceso que está actualmente en ejecución.

Un cambio de contexto puede ocurrir como respuesta a

- Una Interrupción

- Una Excepción
- Una llamada al sistema (system call)

En cualquiera de estos tres casos, el proceso en ejecución será interrumpido y el sistema operativo tomará el control para ejecutar la rutina de manejo de la interrupción, excepción o system call. Veamos algunas definiciones:

Interrupción

Una interrupción es un mecanismo mediante el cual el hardware del sistema se comunica con el sistema operativo. La característica más importante es que las interrupciones ocurren en momentos aleatorios durante la ejecución de un proceso, como mencionamos, en respuesta a señales del hardware. Cabe destacar que desde un proceso también se pueden generar interrupciones.

Algunas de las interrupciones más importantes son:

- **de reloj**: cuando ocurre, el sistema operativo pasa el proceso en ejecución (estado “Ejecutando”) a estado “Listo” cuando agotó su tiempo máximo de uso del procesador.

- **de E/S**: cuando ocurre, el sistema operativo determina si ocurrió algún evento por el cual están esperando 1 o más procesos y en tal caso, mueve todos los procesos en estado “Bloqueado” a “Listo” y los procesos en estado “Bloqueado/Suspendido” a “Listo/Suspendido”. El sistema operativo debe decidir si reanuda la ejecución del proceso que previamente se encontraba en estado “Ejecutando” o si lo expulsa y ejecuta otro proceso en estado “Listo” con mayor prioridad.

Excepción

Las excepciones ocurren cuando el procesador detecta una condición de error mientras ejecuta una instrucción como, por ejemplo, una división por cero. El procesador detecta una variedad de condiciones de error que incluyen protección de memoria, fallos de página y fallas internas de la máquina. En particular, una excepción es muy similar a una interrupción en cuanto a que se interrumpe la ejecución del proceso en ejecución, pero difiere en que una excepción se produce por una condición interna del proceso, mientras que una interrupción se produce por un evento externo al proceso.

El sistema operativo debe decidir si la excepción es irreversible, en cuyo caso el proceso se pone en estado “Saliente” y otro proceso será seleccionado para ser ejecutado.

Las excepciones podemos clasificarlas en:

- **Abort**: es una excepción que no siempre informa la ubicación precisa de la instrucción que causa la excepción y no permite reiniciar el programa o la tarea que causó la excepción. Se utilizan para informar errores graves, como errores de hardware y valores incoherentes o ilegales en las tablas del sistema.

- **Faults**: es una excepción que generalmente se puede corregir y que, una vez corregida, permite reiniciar el programa sin pérdida de continuidad. Cuando se informa de una falla, el procesador restaura el estado de la máquina al estado anterior al comienzo de la ejecución de la instrucción que falla. La dirección de retorno para el manejador de fallas apunta a la instrucción que falla, en lugar de a la instrucción que sigue a la instrucción que falla. Un ejemplo típico de esto es un page fault o fallo de página:

    - fallo de memoria: el procesador se encuentra con una referencia a una dirección de memoria que no se encuentra en memoria principal. El sistema operativo debe cargar el bloque (página o segmento) desde el disco en la memoria principal. Mientras todo esto ocurre el proceso que estaba en ejecución, es pasado a estado “Bloqueado”. Cuando el bloque de memoria es cargado, el proceso pasa a estado “Listo”.

- **Traps**: es una excepción que se informa inmediatamente después de la ejecución de la instrucción que generó el trap. Los traps permiten que la ejecución de un programa o tarea continúe sin perder la continuidad del programa. Es responsabilidad del sistema operativo analizar la naturaleza del trap y decidir si se reanuda la ejecución del proceso que la generó o no. La dirección de retorno del controlador de traps apunta a la instrucción que se ejecutará después de la instrucción que generó el trap. Ejemplos de trap pueden ser: breakpoint, división por cero, intento de acceso a memoria protegida.

Llamada al sistema (system call)

Cuando el proceso en ejecución realiza una llamada al sistema como, por ejemplo, apertura/lectura/escritura de archivos, el sistema operativo toma el control, salva el contexto del proceso en ejecución y ejecuta la rutina en cuestión. Al finalizar dicha rutina, restaura el contexto del proceso que realizó la llamada al sistema para que pueda continuar su ejecución.

Cambio de modo

Como mencionamos anteriormente, cuando ocurre una interrupción, se ejecuta una rutina del sistema operativo que trata dicha interrupción. Ahora bien ¿Qué ocurre si hay un proceso de usuario en ejecución?

Si había un proceso de usuario en ejecución, el procesador funcionaba en ***modo usuario*** y como la rutina de procesamiento de interrupciones es una rutina del sistema operativo, el procesador debe cambiar a ***modo núcleo*** (también llamado **modo kernel**) para poder ejecutar dicha rutina.

Ejecución del sistema operativo

Recuerde	• El sistema operativo es un conjunto de programas ejecutados por el procesador • El sistema operativo con frecuencia cede el control y depende del procesador para recuperar el control

Partiendo de la base de que el sistema operativo es un conjunto de programas (software como cualquier otro) y que depende del procesador para ser ejecutado en conjunto con el resto de los programas, existen 3 formas diferentes de abordar una solución al problema cómo controlar y organizar todos los programas que requieren ejecutarse:

Núcleo sin procesos

En esta solución, el sistema operativo se ejecuta por fuera de todo proceso de usuario, es más, el concepto de proceso tal y como fue detallado sólo aplica a procesos de usuarios. Con este diseño, cada vez que un proceso de usuario es interrumpido se realiza un cambio de contexto completo.

En esta alternativa, el sistema operativo es una única pieza de software separada del resto de los procesos de usuario.

Si volvemos a la unidad anterior, y analizamos los distintos diseños de sistemas operativos, encontraremos que existe una analogía con el diseño monolítico y/o en capas.

Ejecución dentro de los procesos de usuario

Esta alternativa, propone que el sistema operativo se ejecute virtualmente en el contexto del proceso de usuario. Con esta solución, una llamada al sistema se ejecuta dentro del entorno del proceso de usuario sin tener que realizar un cambio de contexto completo.

Cuando ocurre una interrupción, excepción o llamada a funciones del sistema, se salva parte del contexto del proceso y se realiza un cambio de modo del procesador para ejecutar las rutinas del sistema operativo, pero no se realiza un cambio de contexto completo, sino que se realiza un cambio de modo dentro del mismo proceso de usuario.

La ventaja primordial de esta solución, es que una vez que el sistema operativo finaliza su ejecución, si el proceso a restaurar es el mismo, no se incurre en un doble cambio de contexto.

Un concepto que se desprende de esto es que podemos ver a un proceso como un contenedor y que no necesariamente la relación entre proceso y programa es 1 a 1.

Para finalizar este modo de ejecución, podemos encontrar cierta analogía con la descripción que dimos en la Unidad I respecto de los microkernels.

Sistemas operativos basados en procesos

Esta última alternativa propone al sistema operativo como una colección de procesos de sistema. Al igual que en las otras alternativas, el software que forma parte del sistema operativo se ejecuta en modo núcleo. En esta alternativa, las principales funciones del sistema operativo se organizan como procesos independientes. Al igual que en la opción anterior, debe existir una porción del sistema operativo que se encuentre separada de todo proceso que brinde funcionalidad mínima para permitir el intercambio de procesos.

La ventaja de este tipo de solución es que impone una disciplina de diseño de programas que refuerza el uso de sistemas operativos modulares con mínimas y claras interfaces entre los módulos.

En este último caso, veremos que existe una semejanza entre este modo de ejecución y el diseño cliente-servidor de los sistemas operativos, descrito en la Unidad I.

Unidad III

~ Procesos e Hilos ~

"La programación secuencial es realmente difícil, y la programación paralela es un paso más allá de eso".

~ Andrew S. Tanenbaum

Procesos e Hilos

Introducción

En los sistemas operativos tradicionales, cada proceso tiene un espacio de direcciones, recursos asociados y un solo hilo de ejecución. De hecho, esa es casi la definición de un proceso. Sin embargo, hay situaciones en las que es deseable tener múltiples hilos de ejecución en el mismo espacio de direcciones ejecutándose casi en paralelo, como si fueran procesos separados (excepto por el espacio de direcciones compartido).

La pregunta que podría surgir es ¿Para qué necesitamos múltiples hilos de ejecución dentro del mismo proceso?

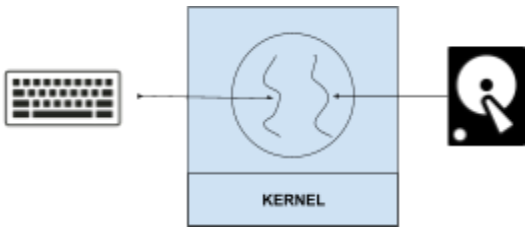
La razón principal es que en muchas aplicaciones necesitamos que varias acciones se ejecuten al mismo tiempo, es decir, que un proceso realice múltiples tareas y que a su vez puedan compartir el espacio de memoria. Lograr esto con múltiples hilos de ejecución dentro del mismo proceso es más simple que tratar de alcanzar la misma solución con procesos separados.

La segunda razón, recae en el análisis de rendimiento. Crear, eliminar o intercambiar un hilo es mucho más rápido que su equivalente operación con procesos, aproximadamente 10 a 100 veces.

La tercera razón tiene que ver también con cuestiones de mejora de rendimiento. Si varios hilos de ejecución realizan muchas operaciones de I/O (entrada/salida), se podrán paralelizar las operaciones aprovechando mejor el uso del CPU.

Algunos ejemplos de aplicaciones que utilizan hilos, pueden ser:

a. Un procesador de texto que realiza 2 tareas básicas. Por un lado, lee los comandos introducidos por el usuario y por otro lanza una tarea de guardado preventivo de los datos. Imaginemos que en caso de no utilizar hilos, podría pasar que el usuario intente utilizar el procesador de texto cuando se está llevando a cabo la tarea de guardado preventivo de datos, por lo que no podría atender la entrada del usuario, que sin lugar a dudas, notaría una baja de rendimiento del programa.



b. Podemos pensar también en un servidor de páginas web, en el cuál se cree un hilo de ejecución para responder a cada pedido de página (request). Imaginemos que este servidor trabaja con una memoria interna similar a una caché donde, cada vez que devuelve una página, la guarda en dicha memoria caché. Un pedido posterior, genera que se cree otro hilo, que puede verificar si la página solicitada se encuentra en el espacio de memoria del proceso (servidor web) y en caso de que se encuentre la devuelve. Por el contrario, si la página no se encuentra en el espacio de memoria del proceso, el hilo en cuestión deberá ir a buscarla al disco. Esto generará que el hilo pase a estado bloqueado y se dará lugar a la ejecución de otros hilos

Otros contextos en los que es conveniente la utilización de hilos son:

- Aplicaciones que trabajen en primer y segundo plano.
- Aplicaciones que proporcionen procesamiento asincrónico.
- Aplicaciones que requieran procesar gran volumen de información y no perder rendimiento.

Cabe destacar que el uso de hilos en aplicaciones es realmente útil, o mejor dicho aprovechable, cuando se dispone de varios núcleos o procesadores.

Definición

Como mencionamos anteriormente, en el enfoque tradicional, tal como lo estudiamos hasta ahora, vimos que un proceso consta de 2 características principales:

1. **Propiedad de recursos**: un proceso incluye un espacio de direcciones de memoria para el manejo de la imagen del proceso (programa, datos, pila y PCB)
2. **Planificación / ejecución**: un proceso puede ser planificado y activado (suspendido, bloqueado, etc) por el sistema operativo.

Tradicionalmente estas 2 características son la esencia de un proceso; pero estas características podrían ser tratadas como independientes una de la otra por el sistema operativo. Esto es lo que hacen muchos (o la mayoría) de los sistemas operativos modernos. Para distinguir entre ambas

características, la unidad que se planifica/activa lo denominaremos **hilo (thread)**, o **proceso liviano**, mientras que la unidad que posee los recursos se suele denominar **proceso.**

Lo que el concepto de hilo añade al modelo ya estudiado de proceso es permitir que se realicen distintas tareas, que podrían ser independientes entre sí, en el mismo espacio del proceso.

Multihilo

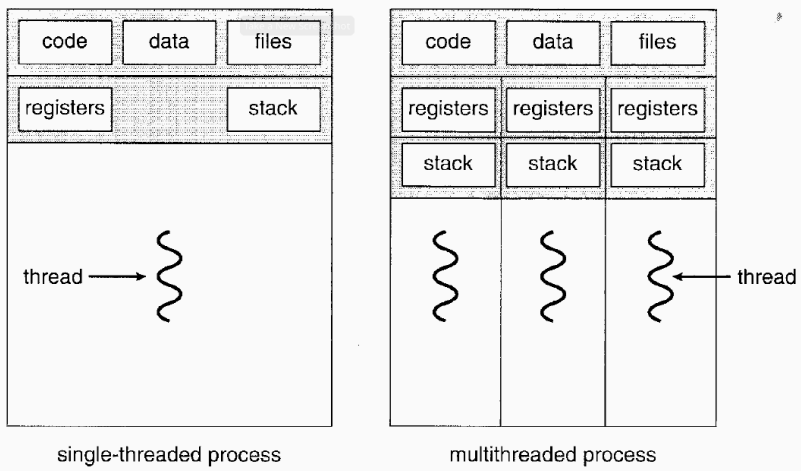
Definición	Es la capacidad de un sistema operativo de dar soporte a múltiples hilos de ejecución en un solo proceso.

Al enfoque tradicional de un solo hilo de ejecución por proceso, se lo conoce como estrategia **monohilo**, en cambio una estrategia multihilo es aquella que soporta múltiples hilos por proceso.

Dentro de un proceso, puede haber uno o más hilos, cada uno con:

- Su código
- Registros (estado de ejecución, pc, etc)
- Una pila de ejecución
- Espacio de almacenamiento para variables locales.
- Acceso a memoria y recursos de su proceso. Esto es compartido con todos los hilos del mismo proceso.

El siguiente gráfico trata de poner de manifiesto las diferencias entre un entorno monohilo y un entorno multihilo:



En un entorno multihilo, un proceso se define como la unidad de asignación de recursos y una unidad de protección. Podemos tener distintos esquemas o configuraciones al respecto:

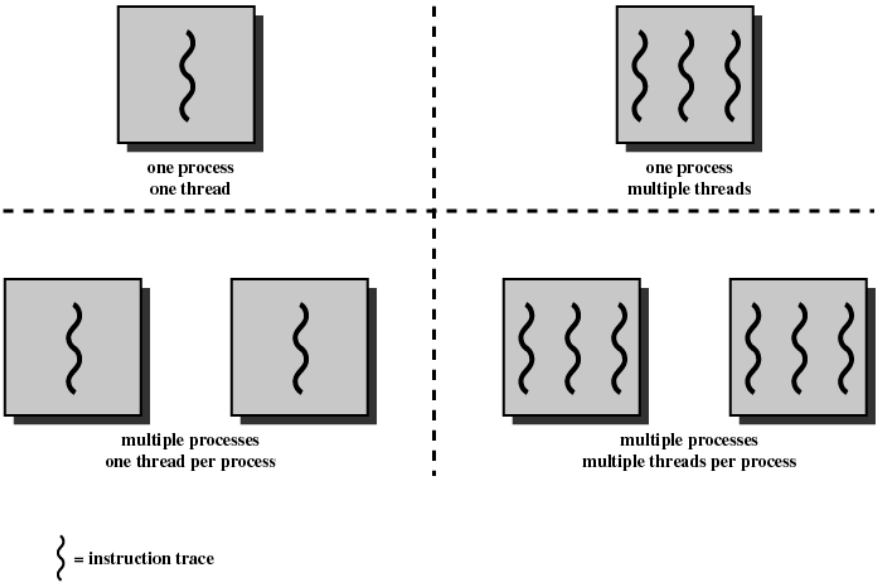


Figure 4.1 Threads and Processes [ANDE97]

Como vemos en la figura anterior, las representaciones que se encuentran a la izquierda de la línea punteada vertical, son procesos con un único hilo, mientras que las representaciones a derecha son procesos con múltiples hilos.

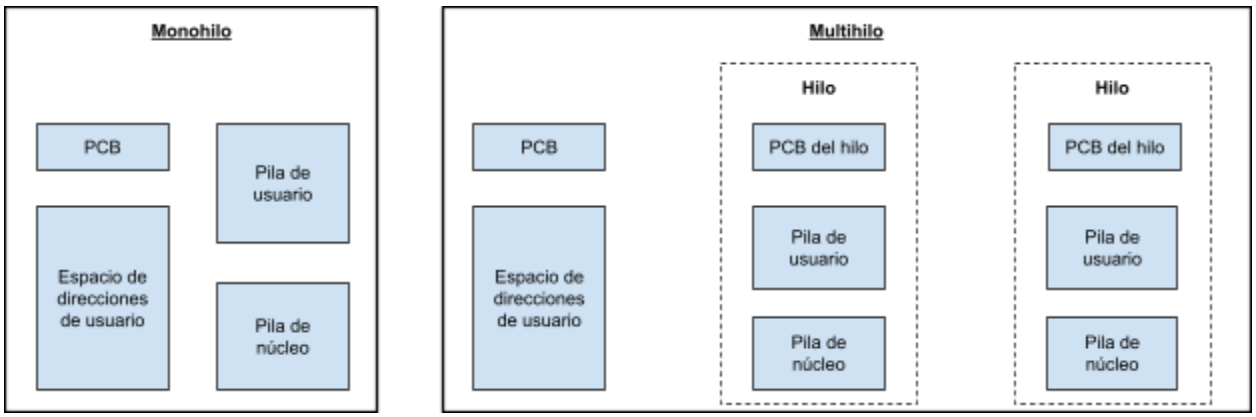
Los sistemas UNIX tradicionales, eran sistemas operativos multiprogramados pero que sólo soportaban un único hilo por proceso (monohilo).

Cuando se ejecuta un proceso con múltiples hilos en un sistema con un único CPU, la ejecución de los hilos entrelaza en el tiempo, similar a lo visto con multiprogramación y procesos. El sistema da la ilusión de procesos/hilos que se ejecutan en paralelo.

Diferencias entre proceso e hilo

En un modelo monohilo, la imagen del proceso incluye el PCB, espacio de direcciones de usuario y la pila de usuario y núcleo para las llamadas y retornos de funciones.

En un entorno multihilo, sigue existiendo un único PCB un un espacio de direcciones de usuario asociado al proceso, pero aparecen múltiples pilas y múltiples PCB uno por cada hilo del proceso.



Del gráfico anterior se observa que todos los hilos de un mismo proceso comparten el estado y los recursos del proceso, residen en el mismo espacio de memoria y tienen acceso a los mismos datos; con lo cual, si un hilo modifica algún dato, todo el resto de los hilos tienen acceso al dato modificado.

De lo expuesto, se deriva que no existe un mecanismo de protección entre hilos como sí lo hay entre procesos independientes. Esto se debe a que:

1. Todos los hilos de un proceso siempre pertenecen al mismo usuario, que es dueño del proceso. Distinto a lo que ocurre con procesos donde cada proceso puede pertenecer a un usuario distinto.
2. Todos los hilos de un proceso comparten el mismo espacio de memoria.

En la siguiente tabla se especifican los ítems compartidos entre los distintos hilos de un mismo proceso y los ítems propios (privados) de cada hilo:

Ítems por proceso	Ítems por hilo
• Espacio de memoria (address space) • Variables globales • Punteros a archivos • Punteros a procesos hijos • Señales • Información estadística	• PC (program counter) • Registros • Stack • Estado

Ventajas de usar hilos

Los mayores beneficios provienen de las mejoras en el rendimiento:

1. Es más rápido crear un nuevo hilo dentro de un proceso existente que un nuevo proceso.
2. Es más rápido finalizar un hilo que un proceso.
3. Es más rápido el cambio entre hilos que entre procesos.
4. Los hilos mejoran la eficiencia de la comunicación entre tareas. La comunicación entre procesos independientes requiere de la intervención del sistema operativo, en un entorno multihilo, al estar los hilos dentro del mismo proceso, la comunicación se puede realizar sin intervención del sistema operativo.

Estados de los hilos

En un sistema operativo multihilo, la planificación se realiza a nivel de hilo. Sin embargo, existen diversas acciones que afectan el estado de todos los hilos de un proceso, por ejemplo suspender o finalizar un proceso implica que todos sus hilos sean suspendidos o finalizados respectivamente.

Los principales estados que se manejan a nivel de hilos son:

- **Creación**: cuando se crea un proceso también se crea un hilo. Luego se pueden crear otros hilos y cada uno tendrá su propio contexto, pila, etc.
- **Bloqueo**: cuando un hilo espera por algún evento, se bloquea, almacenando los registros de usuario. El procesador puede ejecutar otro hilo en estado Listo dentro del mismo proceso o en otro.
- **Desbloqueo**: cuando ocurre el evento por el cual el hilo bloqueado espera, el mismo pasa a la cola de listos.
- **Finalización**: cuando se completa un hilo, se liberan sus registros

No tiene sentido estados como Suspendido, ya que son conceptos que se aplican a nivel de proceso.

Sincronización de hilos

Cuando desarrollamos aplicaciones utilizando hilos, contamos con beneficios en cuanto al rendimiento de la ejecución de nuestro programa, gracias a que parte de nuestra aplicación se ejecutará de forma solapada. Pero la concurrencia también acarrea algunos problemas como son la condición de carrera o el deadlock. Por suerte, existen técnicas que nos ayudan a evitarlos, Veremos estos problemas y posibles soluciones en las próximas unidades.

Hilos en el espacio de usuario o en el espacio del kernel

Existen principalmente dos formas de implementar hilos: una son los llamados hilos en el espacio de usuario y otra son los hilos en el espacio del kernel, aunque también es posible una implementación híbrida.

Cualquiera de las formas mencionadas requieren de una biblioteca que provea las funciones correspondientes para gestionar los hilos.

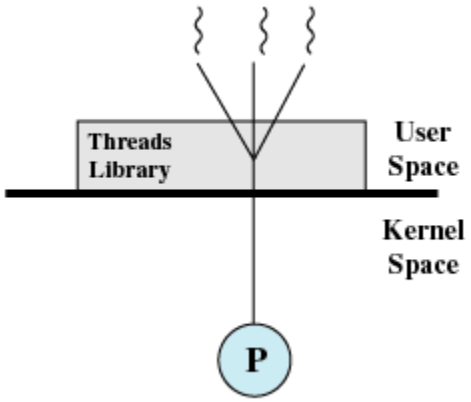
Una biblioteca de hilos proporciona al programador una interfaz para gestionar los hilos. Continuando con lo mencionado en el primer párrafo, dicha biblioteca puede implementarse en el espacio de usuario o puede ser el sistema operativo quien la provea, es decir, implementada a nivel del kernel. La principal diferencia entre estos dos enfoques es que en el primer caso, para gestionar hilos se invocarán funciones dentro del mismo programa, mientras que en el segundo caso serán llamadas al sistema (syscall).

Existen principalmente 3 bibliotecas para manejo de hilos:

1. POSIX Pthreads: puede implementarse tanto a nivel usuario como kernel
2. Win32: sólo disponible en sistemas Windows y a nivel kernel.
3. Java: utiliza la biblioteca del sistema host, por lo que puede utilizar POSIX o Win32.

A continuación describiremos los dos métodos mencionados para implementar hilos, junto con sus ventajas y desventajas.

Hilos en el espacio de usuario o ULT



El enfoque ULT (*User Level Thread*) puro consiste en gestionar los hilos completamente en el espacio del usuario, es decir desde el proceso que los genera. El kernel no gestiona los hilos, de

hecho no conoce su existencia. En lo que respecta al kernel, está gestionando procesos ordinarios de un solo hilo.

La ventaja es que se puede implementar hilos en un sistema operativo que no admite hilos. Esto se logra a través del uso de bibliotecas.

Todos los sistemas operativos tradicionales solían caer en esta categoría.

Por defecto, una aplicación comienza con un solo hilo de ejecución. A partir de ese instante, el proceso puede crear nuevos hilos utilizando llamadas a la biblioteca de hilos. La aplicación cede el control a la biblioteca de hilos y una vez creado el hilo, el control es retornado al hilo principal de ejecución de la aplicación. Es responsabilidad de la aplicación implementar un algoritmo de planificación de hilos para determinar cual de todos los hilos creados, que se encuentren en estado *listo* pasará a ejecutarse.

Como vemos, todo lo descrito sucede dentro del espacio de usuario y dentro de un proceso. El kernel continúa planificando el proceso como una unidad y asigna al proceso un único estado.

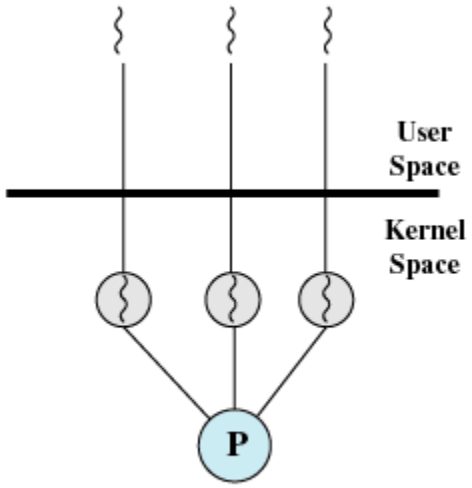
Ventajas de utilizar hilos a nivel de usuario:

1. El cambio entre hilos del mismo proceso no requiere intervención del núcleo, por lo que no se realiza un doble cambio de modo para gestionar los hilos.
2. El algoritmo de planificación puede especificarse dentro de la aplicación.
3. Los ULT son portables, es decir, pueden ejecutarse en cualquier sistema operativo.

Desventaja de utilizar hilos a nivel de usuario:

1. Cuando un hilo realiza una llamada al sistema, si la misma es bloqueante, resulta en el bloqueo del proceso completo.

Hilos en el espacio del kernel o KLT



(b) Pure kernel-level

En un entorno KLT puro, toda la gestión de hilos la realiza el kernel, no hay código de gestión de hilos en el proceso de usuario. Sólo existe una API para acceder a las utilidades del sistema operativo para gestión de hilos.

Al igual que en el enfoque ULT, cualquier aplicación puede programarse para utilizar hilos. Cuando una aplicación necesita crear un hilo, realiza una llamada al sistema, lo que es considerablemente más costoso que en el enfoque ULT debido a que debe intervenir el sistema operativo.

La principal ventaja de este enfoque reside en que si un hilo ejecuta una acción bloqueante, el sistema operativo puede planificar otro hilo del mismo proceso para que comience a ejecutar

Como mencionamos anteriormente, la principal desventaja de utilizar el enfoque KLT en comparación con el enfoque ULT, es que un cambio de hilo del mismo proceso conlleva un cambio de modo.

A modo de resúmen, podemos decir que si bien hay una ganancia significativa entre el uso de ULT respecto de los KLT, si los hilos realizan muchas operaciones que requieren acceso a modo núcleo, esta ganancia no sería tan superior al enfoque KLT.

Unidad IV

~ Concurrencia, exclusión mutua y sincronización ~

"Nada existe por completo solo; todo
está en relación con todo lo demás".

~ Gautama Buddha

Concurrencia, exclusión mutua y sincronización

Introducción

La concurrencia es un tema fundamental en el diseño de los sistemas operativos modernos, principalmente porque hace a la gestión de procesos e hilos. La concurrencia es fundamental en áreas como la *multiprogramación*, el *multiprocesamiento* y el *procesamiento distribuído*.

Un requisito básico de un sistema operativo, para poder administrar procesos e hilos de manera concurrente, es proveer mecanismos que posibiliten la *exclusión mútua*, es decir, impedir a otros procesos el uso de un recurso si ya fue asignado a un proceso en particular.

Principios de la concurrencia

En un sistema multiprogramado y con un único procesador, los procesos se entrelazan en el tiempo para simular la ejecución simultánea, y aunque no se consigue procesamiento paralelo real y el cambio entre cada proceso supone una sobrecarga al sistema, se obtienen importantes beneficios respecto del rendimiento del procesamiento.

Ahora bien, junto con estos beneficios también comienzan a aparecer algunos problemas que se derivan de una característica básica de los sistemas multiprogramados la cual dice que no se puede predecir la velocidad relativa de ejecución de los procesos; ya que ésta depende de la actividad de otros procesos, de la forma en que el sistema operativo maneje las interrupciones y cómo planifique la ejecución de los procesos. A continuación se describen los principales problemas que pueden generarse:

1. **Compartir recursos globales**. Por ejemplo, si 2 procesos utilizan datos ubicados en memoria compartida (una variable global).
2. **Asignación de recursos a varios procesos**, como puede ser un canal de E/S.

Para ilustrar el primer punto imaginemos que existe el siguiente procedimiento:

```
void eco () {
   cent = getChar();
   var_global = cent;
   putchar(var_global);
}
```

Considere también que existen 2 procesos P1 y P2 que hacen uso del procedimiento *eco*. El procedimiento bien podría estar alojado en una porción de memoria compartida ya que va a ser invocado por varios procesos. Ahora bien, imagine que se comienza a ejecutar el proceso P1 y se

realiza la invocación al procedimiento *eco*. Justo en el momento inmediatamente posterior a la invocación de getChar, el proceso P1 es suspendido y el proceso P2 comienza a ejecutarse. El proceso P2 invoca al procedimiento *eco* y finaliza. Cuando el proceso P1 retoma la ejecución, la variable *var_global* posee el valor que fue leído en la ejecución realizada por P2.

El resultado de esto es que el primer caracter se pierde y el segundo se muestra 2 veces.

Para solucionar este problema, el sistema operativo debe poder garantizar que sólo un proceso a la vez pueda invocar al procedimiento *eco,* o lo que sería más correcto decir es que se deben proteger a los recursos globales controlando el acceso a los mismos.

La situación que se describió en el ejemplo anterior, pertenece a los problemas de tipo *condición de carrera.*

Definición	**Condición de carrera**: situación en la cual múltiples procesos leen y escriben un dato compartido y el resultado final depende de la coordinación relativa de sus ejecuciones

Lo expuesto en la definición, trata de reflejar que una condición de carrera sucede cuando varios procesos o hilos acceden al mismo recurso compartido (ej.: una variable global) y el resultado que se almacena en dicho recurso depende del orden de ejecución de los procesos., en otras palabras, el último proceso que accede define el valor del recurso.

La concurrencia nos lleva a que la preocupación principal de un sistema operativo debe ser asegurar que el resultado producido por un proceso sea independiente de su velocidad de ejecución y el accionar del resto de los procesos.

Interacción entre procesos

Para poder entender las soluciones que podemos aplicar a los problemas originados por la concurrencia, debemos antes entender de qué forma se relacionan los procesos. Existen 3 casos:

Procesos que no se perciben entre sí - Competencia

Son procesos independientes que no se pretende que trabajen juntos y que no realizan intercambio de información alguno. A pesar de esto, el sistema operativo debe ocuparse de la competencia por los recursos.

Si bien estos procesos no tienen noción de la existencia del otro, el tiempo de ejecución de un proceso puede verse afectado por la ejecución de otro ya que estos procesos compiten por los recursos compartidos como puede ser una impresora. A estos recursos se los denomina *recursos críticos*.

Para poder administrar el uso de los recursos críticos surge la necesidad de tener un mecanismo que permita llevar adelante dicha tarea. Esta técnica, que llamaremos **exclusión mutua**, debe permitir el acceso controlado a los recursos críticos, es decir, deberá garantizar que sólo 1 proceso a la vez acceda al recurso compartido.

La aplicación de exclusión mutua requiere, sí o sí, intervención del sistema operativo para poder marcar un recurso como bloqueado.

Desde el punto de vista del código, los programas, deberán definir una **sección crítica** la cual es una porción de código donde se hace uso de los recursos compartidos. Es importante que sólo un proceso a la vez pueda acceder a esta sección crítica.

Si bien la exclusión mutua posibilita el acceso a los recursos compartidos, genera 2 posibles problemas:

- **Starvation** (inanición): significa que un proceso nunca va a terminar su ejecución debido a que no obtuvo acceso a los recursos que requería para finalizar su ejecución.
- **Deadlock** (interbloqueo): significa que 2 o más procesos no van a terminar su ejecución debido a que los recursos que requiere uno de los procesos fueron asignados al otro y viceversa.

Ejemplos:

Supongamos que P1, P2 y P3 requieren acceso al procesador. P1 comienza a ejecutarse. En determinado momento y por el algoritmo de planificación que posee el sistema operativo, P3 pasa a hacer uso del procesador. En algún momento P3 es interrumpido y el sistema operativo en vez de asignar el procesador a P2, se lo asigna a P1 nuevamente y así sucesivamente. Puede suceder que P2 nunca consiga acceso al procesador ergo nunca va a ejecutarse. A esta situación se la conoce como *starvation*.

Por otro lado, pensamos que P4 y P5 requieren acceso a la impresora y al disco rígido al mismo tiempo para completar su trabajo. Por una cuestión temporal, el disco fue asignado a P4 y la impresora a P5. Cada proceso queda a la espera del recurso que le falta sin liberar el que ya posee. A esta situación se la conoce como deadlock.

Procesos que se perciben indirectamente - Cooperación por compartición

Estos procesos, si bien no tienen un conocimiento explícito del resto de los procesos a través de su ID de proceso, comparten el acceso a un objeto común como por ejemplo un dispositivo de E/S, memoria compartida, etc.

Estos procesos pueden leer y actualizar datos compartidos, pero se debe garantizar la consistencia de los mismos. Para lograr este objetivo, sólo las operaciones de escritura deben ser mutuamente excluyentes.

Al igual que en el caso anterior, se puede aplicar a cada proceso la definición de una zona crítica para lograr que sólo 1 proceso a la vez actualice los recursos compartidos.

A los problemas mencionados en el caso anterior se le suma el de mantener la coherencia de los datos.

Procesos que se perciben directamente - Cooperación por comunicación

Son procesos capaces de comunicarse entre sí ya que conocen el ID de proceso del otro y son diseñados para trabajar en conjunto. La comunicación proporciona una manera de sincronizar o coordinar las actividades que realizan los procesos.

La comunicación se lleva a cabo mediante el envío de mensajes. Las primitivas para el envío y recepción de mensajes deben ser proporcionadas por el sistema operativo o por el lenguaje de programación utilizado.

Dado que estos procesos se conocen y cooperan entre sí mediante el paso de mensajes, la exclusión mutua no es un requisito de control; sin embargo tanto el *deadlock* como el *starvation* siguen siendo problemas potenciales. Por ejemplo, tendríamos un caso de interbloqueo (deadlock) si tuviésemos dos o más procesos y cada uno está esperando que el otro le envíe un mensaje para operar. En el caso de la inanición (starvation), suponga que se tienen 3 procesos P1, P2, P3. P1 y P2 están intercambiando mensajes, mientras que P3 está bloqueado esperando un mensaje de P1. No se produce deadlock porque los 3 procesos están activos, pero P3 nunca ejecuta, por lo que se considera starvation.

Exclusión mútua

Soporte por Hardware

Para lograr la exclusión mútua por hardware, una solución posible puede ser deshabilitar las interrupciones. Si se deshabilitan las interrupciones justo antes de que un proceso ingrese en su zona crítica, podrá realizar operaciones sin ser interrumpido. Luego, al salir de su zona crítica se deberán habilitar las interrupciones.

Esta solución, si bien es efectiva puede degradar mucho el rendimiento general del sistema ya que se elimina la posibilidad de intercambiar procesos.

Existen otras alternativas que involucran instrucciones especiales del procesador tales como testAndSet y Exchange. Ambas consisten en utilizar una porción de memoria compartida para proporcionar la exclusión mutua.

Ventajas:

- aplicable a cualquier número de procesos
- puede utilizarse para dar soporte a varias secciones críticas

Desventajas:

- **Espera activa**: los procesos en espera para ingresar a su zona crítica, consumen tiempo del procesador.
- **Posible starvation**: la selección del proceso que ingresa a su zona crítica es arbitraria, por lo tanto, podría suceder que un proceso nunca logre ingresar a su zona crítica.
- **Posible deadlock**: si un proceso P1 ingresa a su zona crítica y es interrumpido, puede ser que el proceso P2 elegido para continuar no logre ingresar a su zona crítica. Si además consideramos que el proceso P2 posee mayor prioridad que P1, P1 nunca será seleccionado para ejecutar.

Técnicas por software

Las siguientes son técnicas por software para proporcionar exclusión mutua entre procesos.

Semáforos

La idea principal es que 2 o más procesos pueden colaborar mediante la emisión de señales, de forma tal que un proceso pueda ser obligado a frenar su ejecución en un punto específico hasta recibir la señal que le permita continuar con su ejecución.

Para esto se utilizan estructuras especiales llamadas *semáforos*. Estas estructuras poseen una propiedad que es un valor entero, el cuál puede ser inicializado en un valor negativo o 0 (cero). Tienen definidas sólo 2 operaciones:

- semWait: decrementa el valor del semáforo. Si el valor pasa a ser 0 (cero) o negativo, el proceso que la ejecutó pasa a estado Bloqueado.
- semSignal: incrementa el valor del semáforo. Si el valor pasa a ser positivo, se desbloquea uno de los procesos bloqueados.

Se asume que estas 2 operaciones son atómicas (no pueden ser interrumpidas).

También existe una versión más resumida que son los semáforos binarios o **mutex,** donde el valor del semáforo sólo puede ser 0 y 1.

Ambos tipos de semáforos utilizan una cola para los procesos bloqueados. La política más favorable para este tipo de casos es FIFO, el proceso que lleva más tiempo bloqueado es el primero que se extrae para ejecutar.

Un semáforo que especifica una política para ordenar y administrar los procesos bloqueados se denomina **semáforo fuerte**, mientras que un semáforo que no especifica alguna política, se denomina **semáforo débil.** Los semáforos fuertes son los más convenientes de implementar.

Monitores

Si bien los semáforos permiten alcanzar la exclusión mutua y la coordinación de procesos, su implementación en un programa puede ser difícil y propensa a errores.

Los monitores, por su parte, son una construcción del lenguaje de programación. Sus principales características son:

- A sus datos sólo se puede acceder a través de funciones o métodos propios del monitor. Los datos no son públicos.
- Para poder ejecutar, un proceso debe solicitar al monitor permiso de acceso.
- Sólo se concede permiso de ejecución a un proceso a la vez.

Gracias a estas características, un monitor proporciona exclusión mutua. Es por esto que si se necesita proteger una estructura de datos compartida, la misma puede ser implementada como datos del monitor.

Cuando un proceso requiere el uso del monitor, solicita acceso a través de métodos del monitor. En caso de que no exista ningún proceso ejecutando, al proceso se le concede el acceso y comienza a ejecutar. En caso de que ya exista un proceso en ejecución utilizando el monitor, el proceso se suspende.

Un monitor maneja variables de condición y colas internas para cada variable de condición donde los procesos bloqueados son ingresados a la espera de que ocurra dicha condición.

Si un proceso que está ejecutando detecta un cambio en alguna variable de condición, emite una señal para que los procesos en espera por dicha condición sean desbloqueados.

Paso de mensajes

Cuando los procesos interaccionan entre sí, deben satisfacerse 2 requisitos fundamentales: sincronización y comunicación. Deben ser sincronizados para conseguir exclusión mutua y la comunicación, los procesos cooperantes la utilizan para intercambiar información. La técnica de

paso de mensajes presenta la ventaja de que puede ser implementada tanto en sistemas distribuidos como en sistemas multi o monoprocesador.

Para realizar la comunicación, existen 2 operaciones básicas que son:

- send (destino, mensaje)
- receive (destino, mensaje)

Ambas tienen una versión bloqueante y no bloqueante. Bloqueante significa que el proceso que realiza la operación se bloquea hasta recibir una respuesta del otro proceso o en el caso de la operación send, hasta que se recibe una confirmación de que el mensaje fue recibido correctamente.

Sincronización

En este punto es importante analizar qué le sucede a un proceso después de haber realizado una operación de envío (send) o recepción (receive). Como mencionamos, ambas operaciones poseen una versión bloqueante y no bloqueante y en base a esto hay 3 posibilidades de configuración de los procesos:

- Envío bloqueante, recepción bloqueante: tanto emisor como receptor se bloquean hasta que el mensaje se entrega.
- Envío no bloqueante, recepción bloqueante: sólo el receptor se bloquea hasta recibir el mensaje. Esta es la combinación más utilizada.
- Envío no bloqueante, recepción no bloqueante: ninguno de los procesos se bloquea.

La segunda opción es probablemente la más elegida. Piense en una operación donde se envíen datos a una impresora, el emisor, luego de enviar el mensaje continúa con otras tareas.

Un peligro potencial del send no bloqueante es que ante un error, podría enviar mensajes repetidamente, provocando una caída en el rendimiento del sistema.

Tener en cuenta que el envío no bloqueante, delega la responsabilidad de la confirmación de la recepción en el programador. Los procesos deben implementar mensajes de respuesta para reconocer la recepción de los mensajes.

Respecto de la operación *receive*, la opción más natural sería la bloqueante ya que un proceso que está esperando un mensaje, probablemente requiera de la información del mensaje para operar.

Uno de los posibles problemas de este enfoque, es que si el mensaje nunca llega, el proceso puede quedar bloqueado indefinidamente.

Direccionamiento

Para poder enviar y recibir mensajes, necesitamos indicar a dónde estamos enviando y de dónde estamos recibiendo los mensajes. Vamos a analizar algunas posibilidades:

Direccionamiento directo: en este caso los procesos tanto emisor como receptor conocen el identificador específico del proceso al cual enviar los mensajes y el proceso del cual reciben los mensajes. Esto suele ser utilizado en procesos cooperantes.

Direccionamiento indirecto: en este caso los mensajes son enviados a áreas compartidas que consisten básicamente en colas de mensajes. A estas colas de mensajes se las conoce también como *buzones* o *mailboxes*. De esta forma, unos procesos envían mensajes al buzón y otros toman mensajes desde el buzón.

Este enfoque tiene la ventaja de que desacopla en gran medida emisor y receptor. También posibilita distintos tipos de configuración como ser 1 emisor - 1 receptor, 1 emisor - N receptores, N emisores - 1 receptor, N emisores - M receptores.

En una relación 1 - 1, se establece una comunicación privada entre 2 procesos.

Una relación N - 1, es útil para casos cliente(N) / servidor (1), donde 1 proceso proporciona servicios a otros. En este caso el buzón se lo conoce como **puerto**.

Normalmente los puertos se asocian de forma estática con un proceso, aunque en ocasiones podría ser dinámica.

Formato del mensaje:

Aquí debemos considerar según los objetivos de los proceso si los mensajes serán de longitud fija o variable.

Los mensajes de longitud fija, tienen la ventaja de minimizar la sobrecarga de procesamiento. En un caso extremo donde se transmita gran cantidad de información, la mismo puede estar contenida en un archivo y el mensaje sólo indicará el nombre del archivo.

Por otra parte, si los mensajes son de longitud variable, el mensaje tendrá al menos 2 partes: *Cabecera(header)* y *Cuerpo(payload)*. La cabecera contendrá información sobre el origen y destino, longitud del cuerpo del mensaje, tipo de mensaje, etc. El cuerpo del mensaje contendrá la información propiamente dicha.

Unidad V

~ Fundamentos del interbloqueo ~

"La programación secuencial es realmente difícil, y la programación paralela está un paso más allá de eso".

~Andrew S. Tanenbaum

Fundamentos del interbloqueo (deadlock)

Definición	**Deadlock**: Es el bloqueo permanente de un conjunto de procesos que o bien compiten por recursos del sistema o bien se comunican entre sí.

Un conjunto de procesos está interbloqueado cuando cada proceso se encuentra bloqueado a la espera de un evento (por lo general, la liberación de un recurso) que sólo puede generar otro proceso bloqueado del mismo conjunto.

Todos los interbloqueos involucran necesidades conflictivas que afectan a los recursos de 2 o más procesos.

Tener en cuenta que la aparición de un interbloqueo depende tanto de la dinámica de ejecución como de los detalles de implementación de los programas.

El sistema operativo y las aplicaciones interactúan / consumen recursos que podemos clasificar en recursos reutilizables y consumibles

Recurso Reutilizable

Un recurso reutilizable es aquél que sólo lo puede utilizar de forma segura un proceso en cada momento y que no se destruye con su uso, como por ejemplo el procesador, disco, memoria, archivos, etc.

Por ejemplo, supongamos que el espacio de memoria disponible para los procesos es de 200 Kbytes y tenemos los siguientes procesos concurrentes:

	Proceso A	Proceso B
1	solicita 80 Kbytes	solicita 70 Kbytes
2	...	...
3	solicita 60 Kybtes	solicita 80 Kbytes

Si cualquiera de los procesos llega hasta la línea 3 se producirá un interbloqueo.

Recurso Consumible

Un recurso consumible es aquél que puede crearse (producirse) y eliminarse (consumirse) indefinida cantidad de veces. Por ejemplo las interrupciones, señales, mensajes, etc.

Aunque los recursos puedan ser inagotables, pueden producirse también situaciones de interbloqueo. Por ejemplo, si consideramos la siguiente situación:

	Proceso A	Proceso B

1
2
3
	`receive(mensaje_de_B)`
	`...`
	`send(B, mensaje)`
		`receive(mensaje_de_A)`
		`...`
		`send(A, mensaje)`

Si en el ejemplo anterior, el receive es bloqueante, puede darse una secuencia de ejecución donde A se bloquee esperando un mensaje de B y a su vez B se bloquee esperando un mensaje de A.

La causa del interbloqueo es por lo general un problema de diseño de las aplicaciones, aunque son muy difíciles de detectar. La dificultad en la detección reside en que la aparición de un interbloqueo muchas veces depende de la secuencia de ejecución de las instrucciones.

Condiciones para el interbloqueo

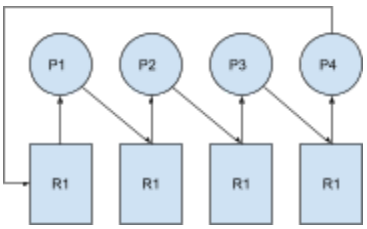
Como mencionamos anteriormente, deben cumplirse ciertas condiciones en el sistema para llegar a la condición de interbloqueo, a saber:

1. **Exclusión mutua**: sólo un proceso puede acceder a un recurso y realizar una acción/función sobre el mismo en un momento determinado.
2. **Retención y espera**: un proceso puede mantener los recursos que le asignaron mientras espera por la asignación de otros recursos.
3. **Sin expropiación**: el sistema operativo no puede forzar la expropiación de un recurso a un proceso que ya lo tiene asignado.

El hecho de que se cumplan estas 3 condiciones (al mismo tiempo), no asegura que exista interbloqueo entre procesos, es decir, estas 3 condiciones son condición necesaria pero no suficiente.

Para que realmente exista interbloqueo entre procesos, hace falta una cuarta condición que llamaremos espera circular.

4. **Espera circular**: existe una lista "cerrada" de procesos, de forma tal que cada proceso posee al menos un recurso necesitado por el siguiente proceso de la lista.



Las 3 primeras condiciones, como dijimos, son necesarias pero no suficientes para la existencia de interbloqueo. La cuarta condición es una potencial consecuencia de las otras tres. Es decir, si se cumplen las 3 primeras condiciones, se puede producir una secuencia de eventos que conduzcan a una espera circular irresoluble.

Esto último es lo que se considera un interbloqueo propiamente dicho: una espera circular irresoluble.

La cuarta condición, la espera circular es irresoluble debido al cumplimiento de las tres primeras condiciones, en resumen, las cuatro condiciones de forma conjunta constituyen las condiciones necesarias y suficientes para la existencia del interbloqueo.

Cabe destacar que las tres primeras condiciones son cuestiones de diseño, mientras que la cuarta condición es una circunstancia que podría ocurrir dependiendo de la secuencia de peticiones y liberaciones realizadas por los procesos involucrados.

Prevención del interbloqueo

Existen métodos de prevención del interbloqueo que podemos clasificar en 2 categorías: los indirectos y los directos.

Un método indirecto trata de impedir la aparición de una de las 3 condiciones necesarias, mientras que un método directo impide que se produzca una espera circular.

Métodos indirectos

Exclusión mutua

En general no puede eliminarse. Si el acceso a un recurso requiere de exclusión mutua, el sistema operativo debe proporcionarlo. Algunos recursos, por ejemplo una base de datos o un archivo pueden tener un acceso compartido (múltiple) si las operaciones son sólo de lectura, pero debe ser exclusivo para las operaciones de escritura.

Retención y Espera

Esta condición puede eliminarse estableciendo que un proceso deba solicitar todos los recursos al mismo tiempo, bloquéandolo hasta que se le puedan asignar los mismos.

Esta solución tiene 2 serios problemas. El primero es que un proceso puede quedar bloqueado mucho tiempo hasta que se le asignen los recursos, cuando en realidad podría haber sido ejecutado sólo con algunos de los recursos. El segundo es que los recursos pueden permanecer ociosos un período de tiempo considerable, durante el cual podrían haber sido usados por otros procesos.

Otra alternativa para evitar la retención y espera, sería permitir a un proceso solicitar un recurso sólo cuando no tiene otro recurso asignado.

Por ejemplo, si un proceso requiere copiar información desde un DVD al disco, luego ordenar el archivo para finalmente imprimirlo en una impresora, según la primer alternativa deberá solicitar los 3 recursos (DVD, disco e impresora) al principio. Esto es un problema de diseño y se consigue tratando de centralizar lo antes posible las llamadas al sistema. Si se implementara la segunda alternativa, el proceso deberá solicitar el uso del DVD, copiar todos sus datos a memoria, luego liberar el DVD y solicitar el disco, copiar los datos al disco, liberar el disco para finalmente solicitar la impresora.

Sin expropiación (no preemption)

Esta condición se puede impedir de al menos 2 formas. La primera es: si a un proceso que ya tiene asignados N recursos, se le deniega la asignación del recurso N+1, deberá liberar los N recursos que ya le habían sido asignados.

La segunda alternativa sería que si un proceso P1 solicita un recurso que fue asignado a otro proceso P2 y P2 se encuentra bloqueado, el sistema operativo puede expropiar los recursos de P2 para asignarlos a P1.

Tener en cuenta que este segundo esquema evita el interbloqueado si y sólo si no existen 2 procesos con la misma prioridad compitiendo por el mismo recurso.

Esta alternativa sirve sólo cuando se aplica sobre recursos cuyo estado puede ser salvado y restaurado con facilidad (como el caso del procesador), no así con una impresora o disco.

Métodos directos

Espera circular

La espera circular se puede impedir definiendo un orden lineal o jerarquía entre los distintos tipos de recursos. Si un proceso solicita y le asignan un recurso R1, luego sólo podrá solicitar un recurso de tipo R2, asumiendo que los recursos forman parte del conjunto R = {R1, R2, ..., Rn}.

Evitar el interbloqueo

Las estrategias analizadas en la *prevención del interbloqueo*, tienden a establecer reglas o restricciones sobre el diseño del sistema operativo y el manejo de los procesos de forma tal que se pueda impedir al menos una de las cuatro condiciones que posibilitan el interbloqueo. La desventaja de estas alternativas es que conlleva a un uso ineficiente de los recursos y una ejecución ineficiente de los procesos.

La estrategia de *evitar el interbloqueo*, por su parte, decide dinámicamente si la petición actual de reserva de un recurso, en caso de que se conceda, puede potencialmente causar un interbloqueo.

Definición	Este tipo de estrategias, requiere conocer de antemano las futuras solicitudes de recursos de cada proceso.

Existen 2 técnicas dentro de la *predicción del interbloqueo*, la primera apunta a no ejecutar el proceso si los recursos que necesita podrían conducir a un interbloqueo y la segunda directamente no asigna los recursos a un proceso si dicha asignación podría provocar un interbloqueo.

Denegación de inicio de un proceso

A partir de la existencia de N procesos y M recursos en el sistema, se definen los siguientes vectores y matrices:

R = {R1, R2, ..., Rm}	cantidad total de cada recurso en el sistema
D = {D1, D2, ..., Dm}	recursos disponibles en el sistema
$N = \begin{matrix} N_{11} & N_{12} & .... & N_{1m} \\ N_{21} & N_{22} & .... & N_{2m} \\ ... & ... & .... & ... \\ N_{n1} & N_{n2} & .... & N_{nm} \end{matrix}$	N_{ij} = necesidades del proceso i con respecto al recurso j

$A = A_{11}\ A_{12}\ ....\ A_{1m}$ $A_{21}\ A_{22}\ ....\ A_{2m}$ $...\ \ ...\ \ ....\ ...$ $A_{n1}\ A_{n2}\ ....\ A_{nm}$	A_{ij} = asignación actual al proceso i con respecto al recurso j

Cada fila de la matriz de necesidad indica los requisitos máximos de cada proceso con respecto a cada recurso.

Se cumplen las siguientes relaciones:

$Ri = Dj + \sum_{i=1}^{n} Aij,\ \forall j$ todos los recursos están o bien disponibles o bien asignados

$Nij \leq Rj,\ \forall\ i,\ j$ ningún proceso puede necesitar más recursos de los existentes en el sistema

$Aij \leq Nij,\ \forall\ i, j$ ningún proceso puede tener asignados más recursos que los necesitados.

A partir de estas relaciones se establece la política de denegación de inicio de un proceso. Un proceso sólo se inicia si se cumple que:

$$Rij \geq N(n+1)j + \sum_{i=1}^{n} Nij,\ \forall\ j$$

Esta relación especifica que sólo podrá iniciarse un proceso si se pueden satisfacer las necesidades máximas de todos los procesos actuales más las del nuevo proceso. El problema de esta política es que asume el peor caso, es decir, que todos los procesos van a solicitar sus necesidades máximas simultáneamente..

Denegación de asignación de un recurso (algoritmo del banquero)

Esta estrategia, también conocida como **algoritmo del banquero**, propone que el sistema puede encontrarse, a partir de la asignación de recursos realizada, en 2 estados posibles: seguro y no seguro.

El estado seguro es aquél en el que hay al menos una secuencia de asignación de recursos a los procesos que no implica un interbloqueo, o lo que es lo mismo decir que todos los procesos pueden ejecutarse por completo.

Un estado no seguro es justamente lo opuesto a un estado seguro.

La idea detŕas de el algoritmo del banquero es que se ejecute cada vez que un proceso realiza una solicitud de recursos. En ese momento el algoritmo determinará en caso de llevar a cabo la

asignación de los recursos solicitados, si el estado del sistema será seguro o no. En caso de que el resultado de la asignación sea un estado no seguro, se denegará la asignación de los recursos al proceso y el mismo se bloqueará hasta que los recursos estén disponibles.

En términos matemáticos, deberá cumplirse la siguiente relación:

$$Nij \ - \ Aij \ \leq \ Dj, \ \forall j$$

Es decir, para que el sistema trabaje en un estado seguro, los recursos necesitados por un proceso, menos los ya asignados deberán ser menores o iguales que los disponibles por el sistema.

Veamos el siguiente ejemplo:

Estado inicial:

	Necesidad			Asignación			Necesidad - Asignación		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	3	2	2	1	0	0	2	2	2
P2	6	1	3	6	1	2	0	0	1
P3	3	1	4	2	1	1	1	0	3
P4	4	2	2	0	0	2	4	2	0

El vector de R (recursos) es R = {9, 3, 6} y el vector D (disponibles) es D = {0, 1, 1}

Con la asignación actual y los recursos disponibles ¿Podemos afirmar que es un estado seguro?

Para responder a esa pregunta debemos averiguar si alguno de los procesos puede ejecutarse por completo con los recursos disponibles. ¿Por qué? Porque si alguno de los procesos puede ejecutarse con los recursos disponibles, una vez que finalice podrá liberar los recursos posibilitando que el resto se ejecute también.

Del estado inicial, vemos que al proceso P2 sólo le hace falta 1 unidad de R3 para poder ejecutarse (eso lo vemos en la 3er matriz). Por lo tanto el sistema operativo puede realizar dicha asignación para que P2 ejecute. Al finalizar la ejecución, P2 liberará los recursos solicitados y pasará a estado finalizado. El estado del sistema resultante será:

	Necesidad			Asignación			Necesidad - Asignación		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	3	2	2	1	0	0	2	2	2
P2	0	0	0	0	0	0	0	0	0
P3	3	1	4	2	1	1	1	0	3
P4	4	2	2	0	0	2	4	2	0

El vector de R (recursos) es R = {9, 3, 6} y el vector D (disponibles) es D = {6, 2, 3}

Note que como P2 finalizó, toda la fila de la matriz N y de la matriz A están en 0 (cero) y el vector D se incrementa.

Ahora tendríamos que repetir las mismas preguntas ¿Podemos afirmar que es un estado seguro? Al igual que antes, si se asignan los recursos al proceso P1, el mismo podrá ejecutar hasta finalizar, dejando el sistema en el siguiente estado:

	Necesidad			Asignación			Necesidad - Asignación		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	0	0	0	0	0	0	0	0
P2	0	0	0	0	0	0	0	0	0
P3	3	1	4	2	1	1	1	0	3
P4	4	2	2	0	0	2	4	2	0

El vector de R (recursos) es R = {9, 3, 6} y el vector D (disponibles) es D = {7, 2, 3}.

El proceso se repite con P3 y P4, determinando entonces que el *estado inicial* planteado era un *estado seguro*.

Ahora bien, suponga el siguiente escenario inicial:

	Necesidad			Asignación			Necesidad - Asignación		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	3	2	2	2	0	1	1	2	1
P2	6	1	3	5	1	1	1	0	2
P3	3	1	4	2	1	1	1	0	3
P4	4	2	2	0	0	2	4	2	0

El vector de R (recursos) es R = {9, 3, 6} y el vector D (disponibles) es D = {0, 1, 1}

Con la asignación actual y los recursos disponibles ¿Podemos afirmar que es un estado seguro?

En este caso está claro que no es un estado seguro ya que no hay ninguna asignación posible que permita satisfacer las necesidades de recursos de un proceso. En este caso todos los procesos requieren al menos 1 unidad de R1.

Detección y recuperación del interbloqueo

Las estrategias para prevenir o evitar un interbloqueo, son muy restrictivas en el sentido de que resuelven el problema limitando el acceso a los recursos o bloqueando procesos. La detección del interbloqueo por su parte es más permisiva ya que los recursos se conceden siempre que sea posible y periódicamente se verifica que no exista la condición de espera circular.

La comprobación de si existe interbloqueo se puede hacer cada vez que un proceso solicita un recurso, o con menor frecuencia aún. Realizar la comprobación por cada solicitud conlleva a una detección temprana del interbloqueo, por otra parte, la desventaja es que se consume un tiempo de procesador considerable.

En caso de que el algoritmo detecte un interbloqueo aplicará alguna de las estrategias de recuperación del interbloqueo que mencionaremos más adelante.

Al igual que en las estrategias anteriores, para posibilitar la detección del interbloqueo se dispone de un vector con la cantidad total de Recursos (R), uno con la cantidad de recursos

Disponibles (D), la matriz de Asignación (A) y una matriz de Solicitudes (S). El algoritmo que posibilita tal detección es el siguiente:

1. Se inicia un vector temporal T asignándole el vector de D.
2. En la matriz A se marcan todas las filas que se encuentren completamente en 0 (cero).
3. De las filas sin marcar en A, se busca el primer proceso cuyas *Solicitudes* de recursos se puedan satisfacer. Es decir, se busca la 1er fila de S que sea menor o igual que D. Si no se encuentran procesos que cumplan tal condición, el algoritmo finaliza.
4. Si se encuentra 1 fila que cumpla la condición del punto 3, se marca el proceso (en la matriz A) y se suma la fila correspondiente al proceso en la matriz A al vector T

Si al final del proceso existen procesos sin marcar, los mismos se encuentran en interbloqueo.

La lógica de este algoritmo es ver qué procesos pueden ejecutarse a partir de las peticiones y recursos disponibles. Esto se realiza en el paso 3, Si se detecta un proceso que puede ser ejecutado, se asume que ejecuta hasta finalizar y libera los recursos asignados. Esto es lo que se hace en el paso 4 al sumar los recursos asignados al vector de recursos disponibles.

Tener en cuenta que a diferencia del algoritmo del banquero, este algoritmo no garantiza la prevención del interbloqueo, sino que determina la existencia actual del interbloqueo.

Ejemplo:

Solicitud

	R1	R2	R3	R4	R5
P1	0	1	0	0	1
P2	0	0	1	0	1
P3	0	0	0	0	1
P4	1	0	1	0	1

Asignación

R1	R2	R3	R4	R5
1	0	1	1	0
1	1	0	0	0
0	0	0	1	0
0	0	0	0	0

El vector de R (recursos) es R = {2, 1, 1, 2, 1} y el vector D (disponibles) es D = {0, 0, 0, 0, 1}

El algoritmo realiza lo siguiente:

1. Establece T = {0, 0, 0, 0, 1}
2. Marca en la matriz A, a P4 porque no tiene recursos asignados
3. Recorre S buscando una fila que sea menor o igual a T. El proceso P3 satisface dicha condición.

4. Actualiza T, sumando el valor de la fila de A correspondiente al índice de la fila encontrada en S. En este caso D = {0, 0, 0, 0, 1} + {0, 0, 0, 1, 0} => D = {0, 0, 0, 1, 1}

El proceso termina porque ni P1 ni P2, es decir ni S[0] ni S[1] son menores que D. Por lo tanto ambos procesos se encuentran en interbloqueo.

Recuperación del interbloqueo

Una vez que se detectó un interbloqueo se pueden aplicar algunas estrategias para recuperar el sistema de dicha situación. En orden de complejidad creciente son:

1. Finalizar todos los procesos que se encuentren participando del interbloqueo
2. Retroceder todos los procesos en interbloqueo a un punto de control previo al interbloqueo. Esta estrategia implica que se implementen mecanismos de retroceso y rearranque. Esta técnica posee el riesgo de que se repita el interbloqueo
3. Finalizar sucesivamente todos los procesos involucrados en el interbloqueo hasta que el mismo desaparezca. Por cada proceso finalizado, se debe invocar al algoritmo de detección.
4. Expropiar los recursos a los procesos involucrados en el interbloqueo hasta que el mismo desaparezca.

Para los puntos 3 y 4 se deben definir políticas de selección que impliquen un costo mínimo. Algunas de estas políticas pueden ser:

a. menor cantidad de tiempo de procesador consumida.
b. menor cantidad de salida producida
c. mayor tiempo restante estimado
d. menor cantidad de recursos asignados
e. menor prioridad

Unidad VI

~ Planificación ~

“La planificación a largo plazo no se ocupa de las decisiones futuras sino del futuro con las decisiones actuales.”

~ Peter Drucker

Planificación del procesador

El objetivo de la planificación es asignar los procesos al procesador para ser ejecutados. La planificación afecta el rendimiento general del sistema porque decide qué proceso ejecutará y qué procesos deberán esperar. Existen 3 tipos de planificación, largo, medio y corto plazo.

Tipos de planificación

Planificación a largo plazo

El planificador a largo plazo determina qué programas son admitidos en el sistema para su procesamiento. Es quien controla el grado de multiprogramación del sistema. Cuando un programa es admitido, se convierte en un proceso y el mismo es añadido a la cola de procesos del planificador a corto plazo.

En algunos sistemas, un proceso recién creado permanece en disco, en cuyo caso se añade a la cola de procesos del planificador a medio plazo.

El planificador a largo plazo toma 2 decisiones, la primera cuando agregar un nuevo proceso al sistema y la segunda qué proceso agregar.

La primer decisión, cuando agregar un nuevo proceso al sistema, se realiza en base al grado de multiprogramación deseado. Tener en cuenta que a mayor cantidad de procesos creados, menor tiempo de procesador tendrá cada uno. Cada vez que un proceso finaliza, el planificador a largo plazo debe decidir si agrega uno o más procesos al sistema. También puede suceder que el tiempo ocioso del procesador supere un determinado valor y se deba invocar al planificador a largo plazo para que incremente la cantidad de procesos en el sistema.

La segunda decisión es la de qué proceso admitir en el sistema, la misma puede ser tan simple como utilizar una política FIFO o ser un algoritmo más complejo que involucre la prioridad, el tiempo estimado de ejecución, requisitos de E/S.

Los estados de los procesos relacionados con el planificador a largo plazo son los estados *nuevo* y *saliente*.

Planificación a medio plazo

La planificación a medio plazo es parte de la función de intercambio (swap), por ende los estados involucrados con este tipo de planificador son los estados: bloqueado, bloqueado/suspendido y listo/suspendido. Este planificador también participa en la decisión de qué grado de

multiprogramación se desea. Para agregar (traer desde disco) un proceso a memoria se deberá contemplar los requerimientos de memoria de los procesos.

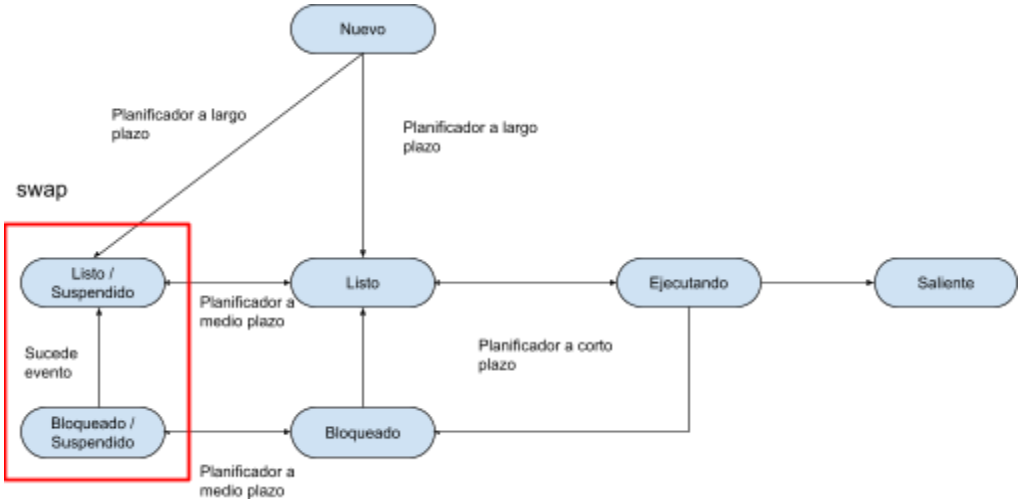
Planificación a corto plazo

El planificador a corto plazo, también llamado activador (dispatcher), es el que toma las decisiones de grano fino sobre qué proceso deberá pasar a utilizar el procesador.

Su objetivo principal es el de optimizar el comportamiento del sistema.

El planificador a corto plazo, es un pequeño programa que intercambia los procesos en el procesador. Se ejecuta siempre que ocurre un evento que puede conducir al bloqueo del proceso actual y que puede proporcionar la oportunidad de expulsar al proceso actual (del procesador) en favor de otro. Estos eventos pueden ser: interrupciones, llamadas al sistema, señales.

Relación entre planificador y los estados de un proceso



Algoritmos de planificación

Criterios

Antes de describir los algoritmos utilizados por el *dispatcher* para planificar la ejecución de procesos, cabe aclarar que existen algunos criterios en los que basarse para evaluar los

algoritmos de planificación. Estos criterios pueden clasificarse en 2 dimensiones, los orientados al usuario y los orientados al sistema.

Los criterios orientados al usuario, se relacionan con el comportamiento del sistema tal y como lo percibe el usuario, mientras que los criterios orientados al sistema se centran en el uso eficiente del procesador.

	Orientado al usuario	Orientado al sistema
Cuantitativo	• **Tiempo de estancia**: tiempo transcurrido desde que se lanza hasta que finaliza el proceso. Es la suma del tiempo de ejecución, tiempo de espera por recursos. • **Tiempo de respuesta**: para un proceso interactivo es el tiempo que transcurre desde que se lanza la petición hasta que se empieza a recibir la respuesta. • **Tiempo límite**: cuando un proceso puede especificar el tiempo límite el planificador debe darle prioridad a dicho proceso y maximizar el número de procesos con tiempo límite terminados.	• **Rendimiento**: apunta a maximizar el número de procesos completados por unidad de tiempo. Es una medida de cuánto trabajo está siendo realizado. • **Utilización del procesador**: es el porcentaje de tiempo que el procesador está ocupado
Cualitativo	• **Previsibilidad**: apunta a que un proceso tarde el mismo tiempo y consuma los mismos recursos independientemente de la carga del sistema.	• **Equidad**: en ausencia de cualquier directiva que oriente la planificación, todos los procesos deben ser tratados de la misma manera, evitando la inanición. • **Imposición de prioridades:** el planificador debe favorecer a los procesos con prioridades más altas. • **Balanceo de recursos**: los recursos del sistema deben permanecer ocupados. Los procesos que utilicen poco los recursos que están sobreutilizados, deberían ser favorecidos.

Los criterios expuestos en la tabla anterior deben ser evaluados a la hora de diseñar un algoritmo de selección de procesos.

Cabe aclarar que estos criterios son independientes y es imposible optimizar todos a la vez. Por ejemplo, disminuir el tiempo de respuesta puede requerir un algoritmo que intercambie procesos con mayor frecuencia, incrementando la sobrecarga del sistema, reduciendo el rendimiento.

Políticas

Cada algoritmo, además de considerar los criterios antes mencionados, implementa una **función de selección** que es la que permite elegir el proceso que debe ser ejecutado. Esta función está basada en alguna política como puede ser la *prioridad*, *requisitos sobre los recursos* o las *características de ejecución del proceso*.

La política de selección también puede contemplar la expulsión de los procesos del procesador.

Políticas sin expulsión: en este tipo de políticas, el proceso que está ejecutando, lo hace hasta que termine o bien hasta que se bloquee esperando por una operación de E/S.

Políticas con expulsión: en este tipo de políticas, el proceso que está ejecutando puede ser interrumpido y pasado al estado *listo* por el sistema operativo.

Las políticas con expulsión, tienen mayor sobrecarga que las no expulsivas, pero pueden proporcionar mejor servicio al sistema en general ya que previenen la monopolización del procesador.

Respecto de las políticas basadas en prioridades de los procesos, cabe aclarar que pueden conducir a una situación de inanición si se aceptan procesos con prioridad más alta que los procesos que están esperando ser ejecutados. Por lo general y para evitar esta situación, los algoritmos que implementan dichas políticas poseen un esquema de modificación de la prioridad del proceso conforme avanza el tiempo.

Primero en llegar, primero en servirse (FCFS)

Este algoritmo implementa un esquema FIFO sobre la cola de procesos. Cuando el proceso que está utilizando el procesador deja de ejecutar (causa de una interrupción, bloqueo, etc) se selecciona para ejecutar el proceso que estuvo más tiempo en la cola de listos.

Este algoritmo proporciona mejor rendimiento en la planificación de procesos largos. Una de las desventajas de este algoritmo es que favorece a los procesos limitados por el procesador sobre los procesos limitados por la E/S.

Definición	Se dice que un proceso está limitado por el procesador si realiza mucho trabajo computacional, es decir, hace un uso intensivo del procesador por sobre el resto de las operaciones. Se dice que un proceso está limitado por la E/S si pasa más tiempo realizando operaciones de E/S que utilizando el procesador.

El hecho de que favorezca los procesos limitados por el procesador se debe en parte a que estos procesos ejecutarán hasta terminar o hasta que se bloqueen por una operación de E/S, mientras que los procesos limitados por la E/S por lo general pasarán más tiempo bloqueados esperando la operación de E/S que usando el procesador. De este análisis también se desprende que tanto el procesador como los dispositivos de E/S estarán por momentos ociosos, desperdiciando tiempo valioso.

Este algoritmo utiliza una política sin expulsión.

Round Robin

Para evitar la penalización que sufren los procesos cortos con FCFS, se puede utilizar un algoritmo como Round Robin basado en las interrupciones de reloj. En este algoritmo cada vez que se produce una interrupción de reloj, el proceso actual es reemplazado por el siguiente en la cola de listos. El proceso que es expulsado vuelve a la cola de listos. La cola de listos se maneja con una política FIFO.

En este tipo de planificación, la clave está en el *quantum* (intervalo de tiempo) a utilizar. Si el quantum es muy chico, habrá una sobrecarga muy grande en el sistema debido a las operaciones de intercambio de procesos. Por otra parte, si el quantum es muy grande, este algoritmo termina pareciéndose a FCFS.

Este algoritmo presenta una desventaja debido a que el tratamiento asignado a los procesos limitados por la E/S no es equitativo con respecto a los procesos limitados por el procesador, ya que estos últimos utilizarán el procesador más veces que un proceso limitado por la E/S. Esto se debe a que los procesos limitados por la E/S estarán bloqueados esperando la operación de E/S mientras que los procesos limitados por el procesador pasarán de *ejecutando* a *listo* y nuevamente a *ejecutando*.

Una alternativa que mejora esta inequidad es el algoritmo Round Robin Virtual en el que se define una cola extra, además de la de *listos*, en la que se colocan los procesos que vuelven del estado *bloqueado*. Al momento de elegir un proceso a ejecutar, los procesos que se encuentren en la nueva cola tendrán prioridad.

Primero el proceso más corto (SPN)

Este algoritmo implementa una política sin expulsión, al igual que FCFS, en el que se selecciona el proceso con el tiempo de procesamiento estimado más corto.

Una complejidad que presenta este algoritmo es conocer el tiempo estimado de procesamiento de un proceso. Este tiempo podría ser indicado por el programador en un sistema por lotes. Este enfoque presenta el problema de que si el tiempo indicado por el programador es muy corto, el sistema expulsará el proceso antes de su ejecución.

En un sistema interactivo puede ser calculado a partir de estadísticas del sistema operativo.

Con este algoritmo, en caso de que exista un ingreso al sistema de procesos cortos, se corre el riesgo de inanición para los procesos más largos y al no posee expulsión, no es un algoritmo adecuado para un entorno de tiempo compartido

Menor tiempo restante (SRT)

Este algoritmo es la versión de SPN pero con una política de expulsión. En este caso el dispatcher elige al proceso con el menor tiempo restante. Cuando un proceso se agrega a la cola de listos podría tener un tiempo restante menor que el proceso que se encuentra actualmente en ejecución. Por lo tanto se producirá la expulsión del mismo, posibilitando la ejecución del proceso nuevo.

A pesar de la política con expulsión, las desventajas de SPN siguen siendo las mismas para este algoritmo.

Feedback

Tanto SPN como SRT requieren conocer el tiempo de servicio de los procesos. Si no se puede obtener, el algoritmo Feedback es una buena alternativa ya que se basa en penalizar a los trabajos que estuvieron más tiempo ejecutando. Esto quiere decir que el algoritmo se basa en el tiempo que cada proceso estuvo ejecutando y no en el tiempo restante.

Para implementar este algoritmo, se definen N colas de prioridad, todas con una política FIFO, desde la 0 a la N donde la cola 0 es la de más prioridad y la N la de menor prioridad. Al igual que en *Round Robin* se define un *quantum* de tiempo para que los procesos utilicen el procesador.

Los procesos ingresan inicialmente a la cola 0 y a medida que van ejecutando y son expulsados van siendo ingresados a una cola de menor prioridad hasta llegar a la cola N.

Cada cola de menor nivel posee un *quantum* de tiempo mayor que la cola inmediatamente anterior.

Otra particularidad de este algoritmo es el tratamiento que otorga a los procesos ante operaciones de E/S. Si el proceso se bloquea por una operación de E/S, al volver del estado bloqueado, es insertado nuevamente en la misma cola en la que estaba antes de bloquearse. Por otra parte si un proceso alcanza la cola de menor nivel y se bloquea ante una operación de E/S, al volver es insertado en la cola inmediatamente superior

Unidad VII

~ Gestión de la memoria ~

“Nunca memorices algo que puedas buscar.”

~ Albert Einstein

Gestión de la memoria

La gestión de la memoria es una tarea fundamental que debe realizar el sistema operativo. En un sistema monoprogramado, la memoria principal sólo se reparte entre el sistema operativo y el programa en ejecución, pero en un sistema multiprogramado, esta división se debe realizar de forma tal que se pueda albergar en memoria la cantidad óptima de procesos de manera que el procesador permanezca ocioso la menor cantidad de tiempo posible.

La gestión de la memoria también es responsable por el intercambio de información entre la memoria principal y la memoria secundaria, básicamente esto es la esencia de la gestión de la memoria.

Requisitos

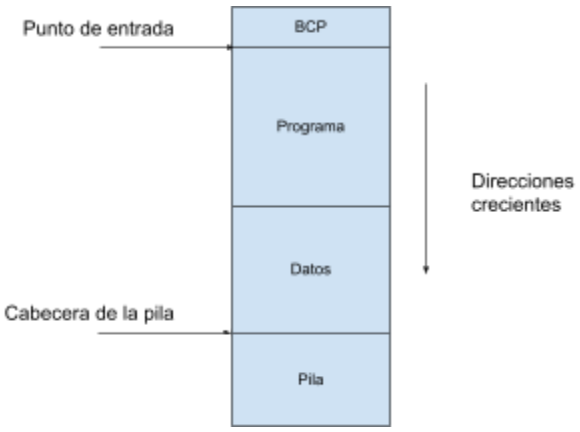
Todo sistema operativo debe satisfacer 5 requisitos de cara a la gestión de la memoria. A continuación los describiremos.

Reubicación

La reubicación es una función sumamente importante que cumple el sistema operativo y que es vital en un sistema que ofrece multiprogramación. La reubicación se encarga de ubicar/alojar un proceso en memoria. Este proceso puede ser o bien nuevo o bien un proceso que fue suspendido y debe ser traído a memoria principal nuevamente.

Pensemos que sin esta función el programador debería conocer anticipadamente en qué porción de memoria debería cargar su programa. Por otro lado hay que tener en cuenta que cuando un proceso es traído nuevamente a memoria, sería muy difícil tener que cargarlo en la misma posición de memoria en la que se encontraba antes de ser suspendido.

El proceso de reubicación, facilita en cierta forma el proceso de traducción de referencias a memoria (instrucciones y datos) de un proceso. Esto se debe a que el encargado de cargar un proceso en memoria principal es el sistema operativo y por tal motivo va a conocer la dirección de memoria donde se ubican el BCP, el punto de entrada del programa y la cabecera de la pila de ejecución del programa. A modo de ejemplo podemos pensar que la imagen de un proceso ocupa una región contínua de memoria como en el siguiente gráfico:



Protección

En un sistema multiprogramado es importante satisfacer el requisito de que ningún proceso pueda acceder al espacio de memoria de otro. Este requisito se complejiza aún más al implementar la reubicación. Por tal motivo, todas las referencias a memoria generadas por un proceso deben ser comprobadas de forma dinámica en tiempo de ejecución.

Los mecanismos de protección, son ejecutados por el procesador (a nivel hardware) ya que sería muy costoso para el sistema operativo calcular y comprobar todas las referencias a memoria de cada proceso y además porque el sistema operativo es otro proceso y no debería controlarse a sí mismo.

Compartición

Los mecanismos de protección deben contar con la flexibilidad suficiente para que si dos o más procesos están compartiendo determinada porción de memoria, puedan acceder a la misma. Por ejemplo, procesos que están cooperando para realizar una tarea podrían tener que acceder a la misma porción de memoria donde se encuentra una estructura de datos.

Organización lógica y Organización física

Tanto la memoria principal, como la secundaria, se organiza como una secuencia unidimensional de bytes o palabras.

Por otra parte, los programas (código + datos) generalmente se organizan en módulos.

El sistema operativo y el hardware del computador deben tratar de forma efectiva los programas de usuario para asignarlos de la mejor manera posible en la memoria. Veremos distintos esquemas, como puede ser la *Segmentación* o la *Paginación*.

Particionamiento

Estático (fijo)

Este es el esquema más simple. Divide la memoria principal en partes fijas, una para el sistema operativo y otra para los programas de usuario. La porción de memoria principal asignada a los programas de usuario, también está dividida en partes fijas.

Particiones de igual tamaño

En este caso, cualquier proceso cuyo tamaño es menor o igual que el tamaño de la partición, puede cargarse en cualquier partición disponible. Si en algún momento todas las particiones se encuentran ocupadas y no hay ningún proceso en estado *Listo* o *Ejecutando* el sistema operativo puede mandar a disco (swap) a un proceso.

La ubicación de un proceso en memoria es un problema trivial debido a que todas las particiones son del mismo tamaño, por lo tanto un proceso que es traído a memoria se ubicará en la primer partición que se encuentre libre.

Esta técnica presenta dos problemas:

- Un programa podría ser más grande que el tamaño de una partición. En este caso el programador deberá utilizar distintas técnicas para cargar en memoria principal el programa.
- Se realiza un uso ineficiente de la memoria principal, ya que sin importar el tamaño de un proceso, siempre ocupa una partición. Esto se conoce como ***fragmentación interna***.

Particiones de tamaño variable

Con particiones de tamaño variable existen 2 formas de ubicar procesos en la memoria principal. La primera y más sencilla consiste en asignar cada proceso en la partición más chica que lo pueda contener. En este caso el sistema operativo mantendrá una cola de procesos por cada partición con el objetivo de que los procesos sean asignados siempre en la misma partición.

La ventaja de este esquema es que reduce la fragmentación interna, pero la desventaja es que podrían quedar particiones de memoria sin utilizar, lo que llevaría el sistema a un uso ineficiente del procesador y una ejecución de procesos poco eficiente.

La segunda alternativa sería tener una única cola de procesos y al momento de cargar un proceso en memoria, se selecciona la partición más pequeña que pueda contener al proceso en cuestión. Nuevamente, si todas las particiones se encuentran ocupadas se podrá seleccionar un proceso para ser enviado a disco (swap). A los algoritmos de planificación que ya mencionamos

en la unidad anterior, podemos agregar que los procesos que ocupen la partición más pequeña que permita almacenar el proceso entrante son los candidatos a ser suspendidos.

Tanto sea un esquema de particiones fijas como uno de particiones variables, las ventajas son su simpleza a la hora de la implementación (respecto de otros esquemas que veremos a continuación) y que presentan una sobrecarga mínima al procesador.

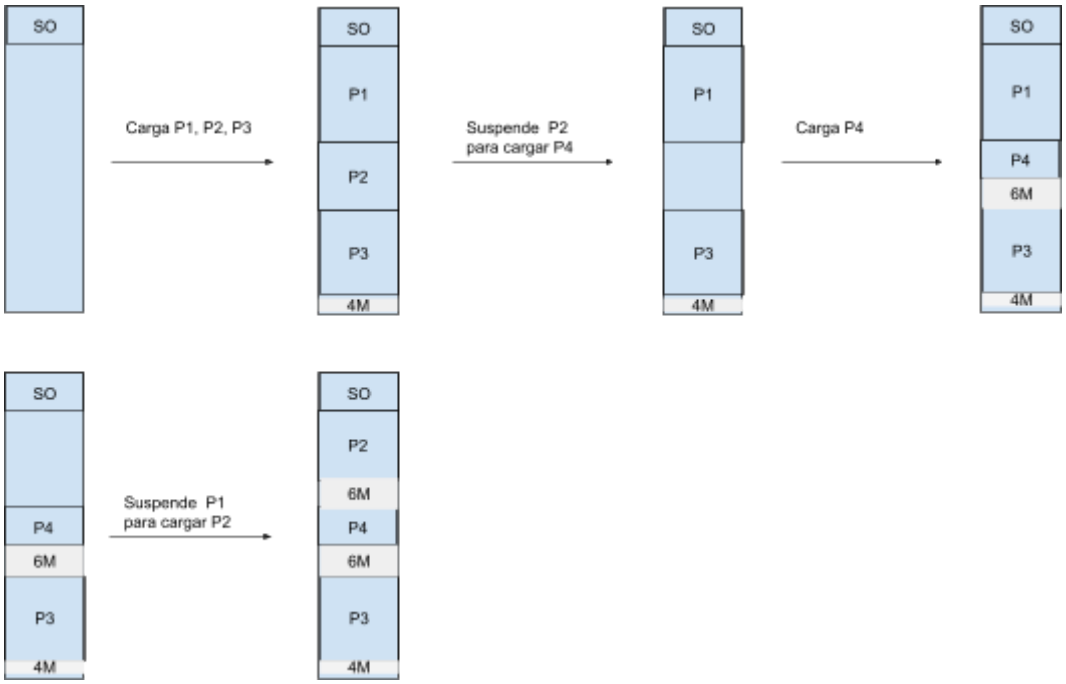
Las desventajas son:

- Se limita la cantidad de procesos activos (no suspendidos)
- El uso de la memoria es ineficiente ya que los tamaños de las particiones se definen en tiempo de compilación del sistema operativo.

Dinámico (variable)

En este esquema las particiones son de longitud y número variable. Cuando un proceso se carga en memoria, se le asigna exactamente la cantidad de memoria necesitada.

Este esquema, si bien parece ser óptimo, presenta la desventaja que genera **fragmentación externa**, es decir, después de cada asignación quedan huecos de memoria que no son asignados a ningún proceso. Veamos esto con un ejemplo: supongamos que tenemos una memoria principal de 64MB. Nuestro sistema operativo ocupa 8M y se necesita ejecutar 4 procesos P1 (20M), P2 (14M), P3 (18M) y P4 (8M).



En el ejemplo vemos que si bien la situación inicial es correcta, el resultado final deja muchos “huecos” en la memoria. A esto se le denomina fragmentación externa.

La diferencia con la fragmentación interna es que la fragmentación interna se produce dentro de una partición, mientras que la externa es sobre la memoria principal.

Para lidiar con este nuevo problema surge la **compactación**. Este proceso lo ejecuta el sistema operativo de forma periódica, reubica los procesos de manera que queden en posiciones de memoria contigua y junta los fragmentos de memoria libre en un solo fragmento.

Algoritmo de ubicación

Debido a que el proceso de *compactación* consume un tiempo considerable, cuando se diseña un sistema operativo se debe pensar bien el algoritmo de ubicación de procesos en memoria. Existen 3 alternativas:

- **Best-fit** o mejor ajuste: selecciona el bloque de memoria que mejor se ajuste al tamaño del proceso a cargar
- **First-fit** o primer ajuste: analiza la memoria desde el principio y escoge el primer bloque libre que sea suficientemente grande para contener al proceso a cargar.
- **Next-fit** o siguiente ajuste: similar a First-fit sólo que analiza la memoria desde la última posición en donde se asignó un proceso.

A modo de resumen podemos decir que ambos esquemas presentan desventajas, por un lado el esquema de particionamiento fijo hace un aprovechamiento ineficiente de la memoria y por otro, el esquema de particionamiento dinámico es más complejo de mantener además de imponer la sobrecarga de la compactación.

Sistema Buddy

Este sistema surge como una alternativa que resuelve las desventajas de los esquemas presentados. Para gestionar la memoria disponible, este sistema divide los bloques de memoria por la mitad hasta obtener un bloque que se ajuste lo mejor posible a la petición de memoria analizada.

El sistema *Buddy* propone que los bloques de memoria disponible son de tamaño 2^k , siendo $L \leq k \leq U$, donde:

- 2^L = bloque de tamaño más pequeño
- 2^U = bloque de tamaño máximo.

El algoritmo empleado para asignar bloques de memoria a un proceso realiza los siguientes pasos:

1. Antes de comenzar se asume que se dispone de un único bloque de tamaño 2^U.
2. Si el proceso a cargar tiene un tamaño *s*, tal que $2^{U-1} < s < 2^U$ se asigna el bloque de tamaño 2^U entero.
3. Si la relación del paso anterior no se cumple, entonces el bloque se divide en 2 bloques de tamaño 2^{U-1}.
4. Si el proceso a cargar tiene un tamaño *s*, tal que $2^{U-2} < s < 2^{U-1}$ se asigna el bloque de tamaño 2^{U-1} entero.
5. En caso de que no se cumpla el paso 4, volver al paso 3.
6. El proceso se repite hasta encontrar el bloque de menor tamaño que pueda almacenar el proceso.

Veamos un ejemplo:

Supongamos que la memoria principal es de 1M y un proceso solicita 100K, el sistema *Buddy* realizará lo siguiente:

1. Se verifica que la relación $512 \leq 100 \leq 1024$ no se cumple, por lo que el bloque de 1M (1024K) se divide en 2 de 512K cada uno.
2. Nuevamente se verifica que $256 \leq 100 \leq 512$no se cumple, por lo que se divide el primer bloque de 512 en 2 de 256.
3. El proceso se repite hasta que se llega a $64 \leq 100 \leq 128$ donde la relación si se cumple, por lo tanto, al proceso se le asigna un bloque de 128K.

<table>
<tr><td>#</td><td colspan="5">Memoria</td></tr>
<tr><td>1</td><td colspan="5">1M</td></tr>
<tr><td>2</td><td colspan="4">512K</td><td>512K</td></tr>
<tr><td>3</td><td colspan="3">256K</td><td>256K</td><td>512K</td></tr>
<tr><td>4</td><td colspan="2">128K</td><td>128K</td><td>256K</td><td>512K</td></tr>
<tr><td>5</td><td>64K</td><td>64K</td><td>128K</td><td>256K</td><td>512K</td></tr>
</table>

¿Cómo es el proceso para recuperar memoria asignada?

Cada vez que un proceso libera memoria que ya no utiliza o el mismo finaliza liberando por completo el bloque de memoria asignada, en la memoria se generan “huecos”, es decir bloques de memoria que no está asignada a ningún proceso.

Cuando otro proceso realice una nueva petición de memoria, el sistema Buddy realizará una búsqueda entre los bloques de memoria libre e intentará realizar la asignación. Si dos bloques libres se encuentran en posiciones contiguas de memoria y la solicitud de memoria realizada, es mayor que uno de los bloques, el sistema unifica dichos bloques en uno solo para poder satisfacer la petición.

Técnicas avanzadas de gestión de memoria

Todos los esquemas vistos hasta ahora son ineficientes desde el punto de vista de la gestión de la memoria y por lo tanto en los sistemas operativos contemporáneos fueron reemplazados por esquemas más sofisticados y eficientes.

Antes de adentrarnos en el análisis de estos nuevos esquemas conviene hacer mención a los distintos “tipos” de direcciones de memoria que maneja el sistema operativo en conjunto con el procesador:

- **Dirección lógica o relativa**: es una referencia a una dirección de memoria y es independiente de la ubicación actual. Debe ser traducida a una dirección física antes de poder ser utilizada. Se dice que es relativa, ya que se define en relación a un punto conocido. Este punto conocido es otra dirección de memoria, normalmente conocida por el procesador, que es el registro base
- **Dirección física o absoluta**: es una dirección real en la memoria principal.

En esquemas de particiones fijas, donde el sistema operativo maneja una cola para cada partición, se espera que cada proceso se cargue siempre en la misma partición a la que fue asignado inicialmente, por lo tanto luego de que el proceso se carga por primera vez, todas las referencias a direcciones de memoria son reemplazadas por direcciones absolutas. Esto se realiza tomando como referencia la dirección de memoria contenida en el registro base del procesador.

En esquemas de particiones fijas, con una única cola para los procesos o esquemas de particiones dinámicos (con reubicación y/o compactación) las direcciones de memoria deben ser traducidas cada vez que el proceso es llevado a memoria.

El proceso de traducción de direcciones de memoria contempla 2 pasos:

1. Primero el procesador traduce la dirección relativa sumando la dirección base del proceso.
2. Verifica que la dirección traducida se encuentre dentro de los límites del proceso.
    a. Si la dirección traducida no se encuentra dentro de los límites del proceso se genera una interrupción, conocida como *segmentation fault* (falla de segmento)

Paginación

Este esquema divide la memoria en particiones de tamaño fijo (relativamente pequeño) denominados **marcos** y cada proceso también es dividido en porciones fijas del mismo tamaño denominadas **páginas**.

Este esquema elimina la fragmentación externa y la fragmentación interna la reduce de tal forma que sólo afecta la última página de cada proceso.

Veamos un ejemplo:

Memoria principal

0	
1	
2	
3	
4	

Carga proc. A y B con 5 marcos libres

A0
A1
B0
B1

Supongamos que A se suspende y para cargar el proceso C necesitamos 3 marcos

0	C0
1	C1
2	B0
3	B1
4	C2

Como vemos las páginas del proceso C pueden cargarse sin necesidad de estar en forma consecutiva.

¿Ahora bien, cómo hace el sistema operativo junto al procesador para resolver una referencia a memoria en un esquema como el graficado?

Un sistema operativo que trabaje con paginación mantiene tablas de páginas por cada proceso. Cada tabla de páginas contiene la ubicación del marco por cada página del proceso, es decir, la dirección de memoria de dicho marco.

Ejemplo:

Proc. A

0	-
1	-

Proc. B

0	2
1	3

Proc. C

0	0
1	1
2	4

Cada tabla de página, contiene el número de marco donde está cargada. Si el proceso se suspende, la tabla es “reiniciada”. Adicionalmente, el sistema operativo mantiene una tabla con los marcos que se encuentran libres.

En este esquema, las direcciones lógicas están expresadas en base al registro base de cada programa y se definen como un número de página más un desplazamiento dentro de la página. Por lo tanto, para traducir una dirección lógica a una física, el procesador deberá:

1. extraer la dirección de la página
2. encontrar cuál es el marco donde se encuentra dicha página.
3. sumar a la dirección del marco el desplazamiento para obtener la dirección física.

Segmentación

Podemos considerar que un programa de usuario se puede dividir en al menos 2 segmentos, uno para el código y otro para datos. Por lo general, estos segmentos son 3, pero 1 de esos segmentos se subdivide en 2:

- **Código**: o segmento de texto, contiene el código ejecutable del programa
- **Datos**:
    - Locales o globales: segmento donde se establecen las variables locales del proceso, así como también las globales y estáticas.
    - heap: segmento donde se establecen las variables cuya reserva de memoria es dinámica. Este segmento puede crecer o disminuir su tamaño.
- **Stack**: segmento utilizado para “apilar” las llamadas a funciones y sus parámetros.

La segmentación ve al espacio de direcciones de memoria de un proceso como un conjunto de segmentos (código, datos, stack). Cada uno de estos segmentos es una unidad lógica que permite al programador organizar el código y los datos de sus programas y a su vez, permite al sistema operativo ubicarlos en porciones de memoria distintas que no necesariamente deben ser contiguas.

La segmentación, al asignar la cantidad de memoria requerida por cada segmento, elimina la fragmentación interna y reduce la fragmentación externa debido a que los tamaños de segmentos son relativamente pequeños.

La segmentación posibilita que varios procesos compartan el segmento de código y el de datos.

Al igual que con la paginación, el sistema operativo crea tablas de segmentos por cada proceso y una lista de bloques de memoria libres.

Para traducir una dirección de memoria de un segmento (lógica) a una dirección de memoria física, se debe recuperar el número de segmento y luego acceder a la tabla de segmentos para recuperar la dirección física donde se encuentra dicho segmento. Finalmente hay que sumar el desplazamiento y así formar la dirección física.

Una vez que se formó la dirección física el procesador verifica que dicha dirección no supere la longitud del segmento. Si así fuera se produce una interrupción conocida como *segmentation fault* o falla de segmento.

Lectura recomendada:

- Segmentation fault

Unidad VIII

~ Memoria virtual ~

“Nunca memorices algo que puedas buscar.”

~ Albert Einstein

Memoria Virtual

Introducción

Las técnicas de paginación y segmentación analizadas en la unidad anterior, presentan dos características importantes que facilitaron la implementación de la multiprogramación:

- Todas las referencias a memoria dentro de un proceso se realizan a direcciones lógicas, que deben ser traducidas a direcciones físicas. Esto posibilita que un proceso pueda ser expulsado de la memoria y reubicado nuevamente en otra posición de memoria.
- Un proceso puede dividirse en porciones y las mismas pueden ser cargadas en posiciones de memoria no necesariamente contiguas.

Ahora bien, por los temas vistos hasta ahora sabemos que para poder ser ejecutado, un proceso debe estar cargado en memoria, es decir, totalmente cargado en memoria.

Aclaración	**¿Cómo sabe el sistema operativo la cantidad de memoria inicial que necesita un proceso para cargarlo en memoria y ejecutarlo?** Esta información se encuentra generalmente en el *header* (encabezado) del archivo ejecutable y la misma es generada en el proceso de compilación.

¿Qué pasa si el espacio ocupado por el proceso es mayor que la cantidad de memoria? Es en este punto donde la memoria virtual se hace presente.

Supongamos que para poder comenzar a ejecutar un proceso sólo es necesario cargar en memoria unos pocos fragmentos[3] iniciales de código y de datos. A este conjunto de fragmentos del proceso, se lo denomina conjunto residente.

Definición	**Conjunto residente**: conjunto de páginas o segmentos de un proceso que siempre están en memoria mientras el proceso se está ejecutando.

Usando una tabla de páginas o de segmentos, el procesador puede determinar si la referencia a memoria se encuentra en memoria principal o no. En caso de que no se encuentre, se genera

[3] Fragmento: página o segmento, depende del esquema utilizado.

una excepción indicando un fallo de acceso a la memoria. El sistema operativo coloca al proceso interrumpido en estado bloqueado y toma el control. El sistema operativo se encarga de cargar en memoria principal el fragmento correspondiente a la dirección lógica que generó el fallo.

Para llevar a cabo este proceso, el sistema operativo realiza una petición de E/S para leer el disco. Mientras dicha operación se completa, otro proceso puede pasar a ejecutarse. Cuando el fragmento solicitado es cargado en memoria principal, se genera una interrupción de E/S para permitir al sistema operativo intercambiar los procesos.

El mecanismo descrito trae aparejado las siguientes ventajas:

1. Pueden mantenerse en memoria principal un número mayor de procesos: esto es posible debido a que sólo se cargan las porciones del proceso que son necesarias.
2. Un proceso puede ser mayor que toda la memoria principal.

Debido a que un proceso sólo se ejecuta estando en **memoria principal**, a la misma se la llama **memoria real**. Pero podemos "percibir" a la memoria potencialmente más grande, utilizando una porción del disco. A esto último se lo denomina **memoria virtual**.

Proximidad y memoria virtual

Supongamos un proceso de gran tamaño, es decir, un programa con mucho código y muchos datos. Si analizamos la ejecución del mismo durante un período muy corto de tiempo, la ejecución se puede acotar a una pequeña sección del programa (por ejemplo una función). Si esto se cumple, sería un desperdicio de la memoria, cargar varios fragmentos del proceso cuando sólo se van a utilizar unos pocos. Entonces cada vez que se produce un fallo de memoria, sería mejor cargar sólo unas pocas porciones del proceso, entre las que se debe encontrar el fragmento deseado.

Esto posibilita que haya más procesos en memoria, y ahorra tiempo debido a que no es necesario intercambiar algunas porciones enviándolas a disco.

Una situación normal es que la memoria se encuentre ocupada con fragmentos de varios procesos, para que el procesador y el sistema operativo tengan acceso a la mayor cantidad de procesos. Cuando se debe traer un fragmento nuevo a la memoria, se debe expulsar otro. Si el sistema operativo expulsa un fragmento justo antes de que vaya a ser utilizado, el mismo deberá ser inmediatamente recuperado. A esto se lo denomina **thrashing.**

Definición	**Thrashing**: el sistema consume la mayor parte del tiempo reemplazando porciones de memoria en lugar de ejecutar instrucciones.

Lo que hace el sistema operativo es tratar de predecir, en base a estadísticas de uso, qué porciones de memoria se deben cargar.. Esta predicción es posible gracias al **principio de proximidad**.

El principio de proximidad indica que las referencias a instrucciones y a los datos dentro de un proceso tienden a agruparse. Por lo tanto, durante un período corto de tiempo, sólo se necesitarán unos pocos fragmentos del proceso.

Paginación

A diferencia de lo visto en paginación sencilla (sin memoria virtual), donde toda la tabla de páginas de un proceso debía ser mantenida en memoria principal, en un sistema que utiliza memoria virtual la tabla de páginas de un proceso no necesariamente debe residir completamente en memoria principal, es decir que algunas páginas también residen en memoria virtual.

Por otra parte, la tabla de páginas de cada proceso modifica su estructura. Se agregan dos bits:

- **P (presente)**: indica si la página se encuentra en memoria.
- **M (modificado)**: indica si la página se modificó desde la última vez que fue cargada. Este flag es útil, llegado el momento de reemplazar la página, para saber si la misma debe ser escrita en disco antes de ser reemplazada.

Esquemas de Tabla de páginas

Dos Niveles

Algunos procesos pueden ser excesivamente grandes, por lo que su tabla de páginas completa no puede mantenerse en memoria principal, sino que debe ser dividida entre la memoria principal y la memoria virtual, en otras palabras debemos paginar la tabla de páginas.

Para manejar esta situación, algunos sistemas implementan un esquema de 2 niveles, en el cual se maneja por cada proceso una tabla llamada “directorio”. El directorio posee una entrada por cada página de la tabla de páginas del proceso.

En un segundo nivel aparece la tabla de páginas de cada proceso donde se relaciona el marco de la memoria principal con cada página. Normalmente el directorio de páginas se diseña para que ocupe lo mismo que una página. El directorio de páginas es lo único que permanece constantemente en memoria hasta que el proceso finaliza.

La ventaja de este esquema es que permite reducir el tamaño de la tabla de páginas de un proceso.

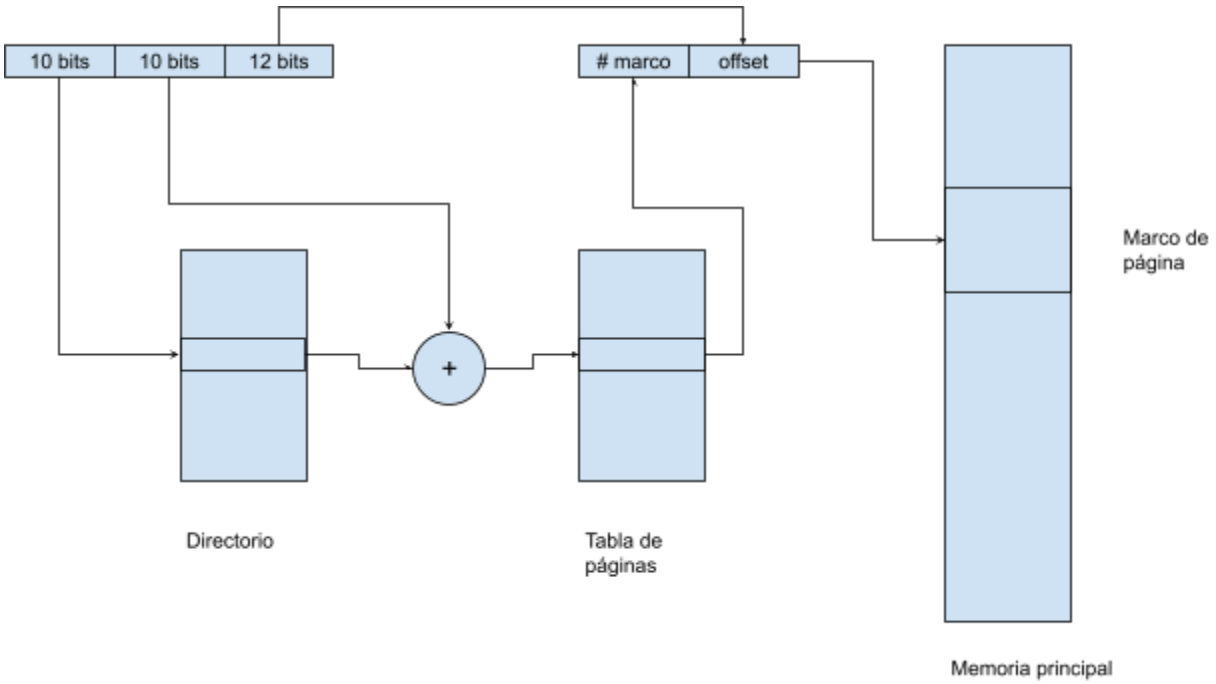
Analicemos esto mediante un ejemplo. Supongamos un sistema con un espacio de memoria direccionable de 32 bits y una página de 4 Kbytes (2^{12} bytes).

Con esos valores, la tabla de páginas de un proceso podría tener 1M de entradas aproximadamente (2^{32} / 2^{12} = 2^{20} = 1.048.576). Manejar una tabla de 1M de entradas es costoso, por lo tanto, si la dividimos en 2 niveles podemos tener una solución de mejor rendimiento y aprovechamiento de la memoria.

Para mejorar esto pensemos lo siguiente:

- Cada dirección de memoria tiene un tamaño de 4 bytes (8 bits * 4 = 32 bits).
- Con un tamaño de página de 4 Kbytes (2^{12} bytes) la tabla de páginas de 1 proceso puede tener hasta 2^{20} páginas, que en memoria ocupan 4 Gbytes (2^{32} bytes).
    - Si cada una de estas 20 páginas, se referencia por una dirección de 4 bytes, la tabla de páginas de un proceso ocuparía 4 Mbytes (2^{20} * 2^2) y para cargar toda la tabla de páginas en memoria (las 2^{20} páginas) necesitamos 2^{10} bytes (2^{22} / 2^{12} = 2^{10} bytes).
- Ahora bien, podemos pensar en una tabla llamada “directorio” de páginas con 2^{10} entradas, una por cada página de la tabla de páginas. Como cada dirección de memoria ocupa 4 bytes, para almacenar 2^{10} entradas, el directorio requiere un espacio de 2^{12} bytes (2^2 * 2^{10}) que equivale al tamaño de 1 página.

Del último punto se deduce que el tamaño del directorio es igual al tamaño de 1 página y llegamos a la conclusión de que en memoria sólo tendremos ocupado 1 página (directorio) más las páginas de la tabla de páginas que se necesiten cargar en memoria para ejecutar el proceso.



Del gráfico anterior vemos que para un espacio de direcciones de 32 bits, una dirección de memoria se conforma de 10 bits para indexar dentro del directorio, 10 bits para indexar dentro de la tabla de páginas del proceso. A partir de ahí se consigue el número de marco al cual se le suma el desplazamiento para obtener la dirección física.

A pesar de tener la ventaja de reducir el tamaño de la tabla de páginas, su principal desventaja es que la cantidad de accesos a memoria puede reducir el rendimiento del sistema.

Lecturas recomendadas

- Paginación en 2 o más niveles. Disponible en: https://www.geeksforgeeks.org/two-level-paging-and-multi-level-paging-in-os/
- Paginación multinivel. Disponible en: https://www.geeksforgeeks.org/multilevel-paging-in-operating-system

Tabla de páginas invertida

En este esquema hay una entrada en la tabla de páginas invertida por cada marco de página real en lugar de por cada página virtual.

La parte correspondiente al número de página de la dirección virtual se referencia por medio de un valor hash.

En este esquema la tabla de páginas siempre requiere una porción fija de la memoria real, independientemente del número de procesos o páginas.

Los campos principales de la tabla de páginas son:

- el Número de Página
- el Id de proceso
- bits de control.

El proceso de traducción de una dirección lógica consiste en:

1. Obtener el número de página de la dirección de memoria y junto con el id de proceso llamar a la función que genera el hash.
2. Con el hash generado acceder a la tabla de páginas invertida para obtener el marco

Buffer de traducción anticipada (TLB)

En principio toda referencia a memoria virtual involucra 2 accesos a memoria, uno para buscar la entrada en la tabla de páginas y otro para buscar los datos.

Para evitar esta situación, la mayoría de los esquemas de memoria virtual implementan una memoria cache especial de alta velocidad para las entradas de la tabla de página. Esta cache se denomina buffer de traducción anticipada (Translation Lookaside Buffer) y contiene aquellas entradas de la tabla de páginas que han sido usadas recientemente.

El proceso de traducción de una dirección lógica es como sigue:

1. El procesador busca la entrada en la tabla de páginas en la TLB.
    a. En caso de encontrarse se recupera el número de marco y se construye la dirección real.
2. Si no se encuentra la entrada en la TLB (fallo de TLB), el procesador accede a la tabla de página. Se recupera el marco y se construye la dirección real.
    a. La entrada de la tabla de páginas accedida será cargada en la TLB, para futuras referencias.
3. Si la referencia no se encuentra en memoria, se produce un fallo de página y el control es cedido al sistema operativo para que cargue la página solicitada.
    a. La entrada de la tabla de páginas accedida será cargada en la TLB, para futuras referencias.

Tamaño de la página

Otro aspecto importante a considerar en cuanto al diseño de un sistema de memoria virtual es el tamaño de cada página.

Este aspecto es importante debido a que a mayor tamaño, menor fragmentación interna. Por otra parte, un tamaño de página grande hace que el *principio de proximidad* pierda fuerza por que habrá referencias a memoria no utilizadas recientemente con lo cual los fallos de página serán frecuentes. Tener en cuenta que si hiciéramos crecer la página a un tamaño tal que pueda contener todo un proceso, no habría fallos de página ni fragmentación interna.

Con un tamaño menor de página habrá muchas más páginas en memoria donde cada una contendrá referencias recientes. Las desventajas son que las tablas de páginas deberán ser más grandes, por lo que habrá mayor cantidad de páginas en memoria virtual. Todo este conjunto de cuestiones nos conduce a la posibilidad de que para una referencia a memoria ocurran 2 fallos de página: el primero por no encontrarse la página de la tabla de páginas y el segundo por no encontrarse la página en memoria principal.

Paginación y segmentación

En un esquema combinado, un proceso se divide en un número de segmentos, donde cada segmento es dividido en un número de páginas de tamaño fijo. Estas páginas son del tamaño de los marcos de la memoria principal.

Desde el punto de vista del programador, una dirección lógica está formada por un número de segmento y un desplazamiento. Desde el punto de vista del sistema, el desplazamiento dentro del segmento es visto como un número de página y un desplazamiento.

Asociado a cada proceso existe una tabla de segmentos y varias tablas de páginas, una por cada segmento.

Políticas del sistema operativo

En todo sistema de administración de memoria se debe evaluar los siguientes 3 puntos:

- ✓ Uso de memoria virtual o no.
- ✓ Paginación, segmentación o ambas
- ✓ Algoritmos utilizados por el sistema operativo para la gestión de la memoria.

Respecto de este último punto, debemos considerar:

1. Políticas de recuperación
2. Políticas de ubicación.
3. Políticas de reemplazo
4. Gestión del conjunto residente
5. Políticas de limpieza
6. Control de carga

Describiremos cada una de estas políticas a continuación.

Políticas de recuperación

Involucra decisiones de cuándo una página debe ser traída a memoria principal. Las 2 alternativas son:

- Bajo demanda: una página se trae a memoria sólo cuando se la referencia. Con esta política cuando un proceso inicia habrá una ráfaga de fallos de página que se irá normalizando a medida que se carguen las páginas referenciadas. Gracias al principio de proximidad los fallos de página tenderán a reducirse
- Paginación adelantada: además de traer a memoria la página buscada, se traen otras. Si las páginas de un proceso se encuentran almacenadas en posiciones contiguas (en memoria secundaria) será más eficiente traer a memoria un número de páginas contiguas de una vez que traerlas de a una.

Políticas de ubicación

Esta política determina en qué parte de la memoria principal van a residir las porciones de la memoria de un proceso. Para sistemas de paginación pura o segmentación más paginación este tema es irrelevante.

Políticas de reemplazo

Cuando todos los marcos de la memoria principal están ocupados, y es necesario traer una nueva página para resolver el fallo de página, esta política determina qué página de las que actualmente se encuentran en memoria va a reemplazarse.

El objetivo es seleccionar la página que tiene menos posibilidades de ser referenciada en un tiempo corto.

Antes de ver los algoritmos utilizados, vamos a comentar que todas las políticas de reemplazo tienen una restricción: algunos marcos de la memoria están bloqueados. Cuando un marco está

bloqueado, la página almacenada en el mismo no puede ser reemplazada. Por ejemplo la mayor parte del núcleo del sistema operativo se carga en marcos bloqueados, buffers de E/S.

Algoritmos básicos

Óptimo

El concepto tras este algoritmo es seleccionar para reemplazar la página para la cuál el instante de la siguiente referencia se encuentra más lejos. Este algoritmo produce la menor cantidad de fallos de página. Este algoritmo es imposible de implementar debido a que el sistema operativo debería conocer los eventos futuros.

LRU

La política *usada menos recientemente* (LRU) seleccionará como página candidata aquella que no se haya referenciado por mucho tiempo. Debido al principio de proximidad, esa página será la que tiene menos probabilidades de volver a ser referenciada en un futuro próximo.

Si bien esta política arroja buenos resultados, similares al algoritmo óptimo, su implementación es muy difícil ya que deberíamos "etiquetar" de alguna forma las páginas con el momento en el que son accedidas para de esa forma realizar la búsqueda al momento de necesitar reemplazarlas.

FIFO

Esta política trata los marcos de página como si fuese un buffer circular y las páginas se reemplazan mediante una estrategia cíclica de tipo *round robin*. Es una de las políticas más simples de implementar. La idea tras este algoritmo es muy simplista, asume que una página traída a memoria hace mucho tiempo, puede haber dejado de utilizarse. Este concepto es erróneo debido a que hay páginas que son cargadas al inicio del programa y referenciadas todo el tiempo. En este caso, esas páginas son expulsadas y traídas de vuelta a la memoria todo el tiempo.

Si bien este algoritmo es muy simple de implementar, su rendimiento es muy pobre.

Reloj

Al igual que FIFO, utiliza un buffer circular de páginas, pero a cada marco de página se le agrega un bit adicional denominado *bit de usado*. Cuando una página se trae a memoria, el bit de usado se pone en 1 y en cualquier momento en que la página se referencie, también se pone en 1 el bit de usado.

Cuando se produce un fallo de página, el sistema operativo recorre el buffer circular de marcos de páginas, buscando aquel cuyo bit de usado esté en 0 (cero). Cada vez que encuentra un

marco con el bit de usado en 1 lo cambia a 0 (cero) y continúa con el marco siguiente. Tener en cuenta que este algoritmo deja situado el puntero en el marco siguiente al reemplazado, de esta forma en el próximo fallo de página se comenzará el algoritmo a partir de ese marco.

Algunos ejemplos:

Flujo de páginas	2	3	2	1	5	2	4	5	3	2	5	2
OPT	2	2	2	2	2	2	4	4	4	2	2	2
		3	3	3	3	3	3	3	3	3	3	3
				1	5	5	5	5	5	5	5	5
					fallo		fallo			fallo		

Flujo de páginas	2	3	2	1	5	2	4	5	3	2	5	2
LRU	2	2	2	2	2	2	2	2	3	3	3	3
		3	3	3	5	5	5	5	5	5	5	5
				1	1	1	4	4	4	2	2	2
					fallo		fallo		fallo	fallo		

Flujo de páginas	2	3	2	1	5	2	4	5	3	2	5	2
FIFO	2	2	2	2	5	5	5	5	3	3	3	3
		3	3	3	3	2	2	2	2	2	5	5
				1	1	1	4	4	4	4	4	2
					fallo	fallo	fallo		fallo		fallo	fallo

Flujo de páginas	2	3	2	1	5	2	4	5	3	2	5	2
RELOJ	2^0	2^0	2^0	2^0	5^0	5^0	5^0	5^0	3^0	3^0	3^0	3^0
		3^0	3^0	3^0	3^1	2^0	2^0	2^0	2^1	2^0	2^1	2^0
				1^0	1^1	1^1	4^0	4^0	4^1	4^1	5^0	5^0
					fallo	fallo	fallo		fallo		fallo	

Reloj mejorado

Este algoritmo es igual a su antecesor, sólo que además del bit de usado agrega un *bit de modificado* el cual se pone en 1 si la página dentro del marco en cuestión fue modificada desde que se trajo a memoria.

Con este nuevo bit, podemos tener las siguientes combinaciones sobre los marcos y sus páginas:

	Usado	Modificado
No se accedió recientemente ni se modificó	0	0
No se accedió recientemente pero se modificó	0	1
Se accedió recientemente pero no se modificó	1	0
Se accedió recientemente y se modificó	1	1

El algoritmo de reemplazo se modifica como se indica a continuación:

1. Comenzando por la posición actual del puntero, recorremos el buffer de marcos. No se cambia ningún bit de usado. El primer marco con usado = 0 y modificado = 0 se selecciona para reemplazo.
2. Si el paso 1 falla, se busca el primer marco usado = 0 y modificado = 1. A medida que se recorren los marcos se cambia el bit de usado a 0 (cero)
3. Si el paso 2 falla, el puntero regresó al marco inicial y todos los marcos tienen el bit de usado en 0 (cero). Se repite el paso 1, con lo que se debe encontrar un marco.

La ventaja de este algoritmo es que da preferencia para el reemplazo a las páginas que no fueron modificadas, de forma tal que se evita una escritura en disco.

Buffering de páginas

Este algoritmo guarda ciertas similitudes con el algoritmo FIFO, sólo que para mejorar su rendimiento se definen 2 listas: una para páginas modificadas y otra para páginas no modificadas. Ante un fallo de página, este algoritmo no elimina inmediatamente de la memoria una página sino que la asigna a la lista que corresponda.

En todo momento este algoritmo se reserva un conjunto de marcos libres sobre los cuales implementa las listas de páginas mencionadas.

Cuando se produce un fallo de página, si la página a reemplazar no fue modificada, se mueve a la lista de páginas no modificadas. Por el contrario, si la página fue modificada se mueve a la lista de páginas modificadas. Note que la página si se elimina de la tabla de páginas del proceso, pero se mantiene en memoria. Esto da la ventaja de que si la página vuelve a ser referenciada no hay que ir a buscarla a disco.

Éstas listas de páginas funcionan como una cache de páginas. Cuando se deben escribir páginas a disco, se escriben todas las páginas al mismo tiempo. Esto reduce de forma significativa el número de operaciones de E/S y por lo tanto los tiempos de acceso a disco.

Gestión del conjunto residente

Esta política aborda las siguientes cuestiones:

1. ¿Cuántos marcos de página se van a reservar para cada uno de los procesos activos?
2. El conjunto de páginas que se va a considerar para realizar el reemplazo se limita a aquellas del mismo proceso que causó el fallo de página o se consideran todos los marcos de páginas de la memoria principal.

Para el primero de los puntos se analizan 2 alternativas, asignación fija y asignación variable.

La asignación fija determina un número fijo de marcos de página para cada proceso, que se determina al momento de cargar el proceso en base al tipo de proceso o a guías proporcionadas por el administrador de sistema. Con la política de asignación fija, cada vez que se produce un fallo de página, la página que se necesite reemplazará a una página del proceso.

La asignación variable permite que el número de marcos del proceso cambie durante el tiempo de ejecución del mismo. Si un proceso está generando muchos fallos de página, el sistema operativo podría asignarle marcos de página extra para disminuir la tasa de fallos de página, mientras que un proceso con una tasa de fallos de página baja, se podrá reducir la cantidad de marcos asignados.

El segundo punto trata el concepto de ámbito de reemplazo, que puede ser local (al proceso) o global (todos los procesos).

Una política de reemplazo local sólo podrá seleccionar páginas residentes del proceso que generó el fallo de página, mientras que una política global considerará todas las páginas que se encuentran en memoria principal independientemente del proceso al que pertenezcan.

Resumiendo, combinando los 2 conceptos sólo podemos tener los siguientes esquemas:

- Asignación fija, ámbito local

- Asignación variable, ámbito global
- Asignación variable ámbito local.

Políticas de limpieza

Esta política determina cuando una página que ha sido modificada debe ser escrita nuevamente en disco. Existen 2 alternativas: limpieza bajo demanda y limpieza adelantada.

Con *limpieza bajo demanda* una página modificada sólo es escrita en disco cuando fue seleccionada para ser reemplazada. Si bien esta técnica minimiza las escrituras de páginas obliga al proceso que produjo el fallo de página a esperar que se completen 2 operaciones de E/S, 1 la de escritura de la página a reemplazar y 2 la carga en memoria principal de la página solicitada.

Por otra parte, con *limpieza adelantada* se escriben las páginas modificadas antes de que sus marcos de páginas se necesiten. La ventaja de esta técnica es que permite la escritura en lotes de las páginas modificadas minimizando las operaciones de E/S.. La desventaja es saber y balancear qué páginas conviene escribir en disco, ya que no tiene sentido escribir una página en disco si va a ser modificada nuevamente antes de su reemplazo.

Una buena alternativa a los problemas de ambas políticas es implementar el algoritmo de *buffering de páginas*.

Control de carga

El control de carga determina el grado de multiprogramación del sistema, es decir, determina el número de procesos que residirán en la memoria principal.

Este aspecto es clave para una gestión efectiva de la memoria. Si la cantidad de procesos es muy baja, hay más posibilidades de que en un momento determinado todos los procesos estén bloqueados y el procesador ocioso. Si por el contrario, la cantidad de procesos es muy alta el tamaño del conjunto residente no será adecuado y los fallos de página serán frecuentes.

El grado de multiprogramación puede incrementarse o reducirse.

Estudios realizados demuestran que si se logra igualar el tiempo medio entre los fallos de páginas al tiempo que demora procesar un fallo de página, entonces se logrará una utilización máxima del procesador. Este estudio fue enunciado por Denning y se lo conoce como L = S.

Otra alternativa es adaptar el algoritmo del reloj para que utilice un ámbito global de selección de páginas. La modificación hace que se monitoree la tasa de fallos de página. Si la tasa está por debajo de un nivel determinado entonces el grado de multiprogramación puede incrementarse

con seguridad. Por el contrario, si la tasa de fallos de página supera dicho nivel, entonces el grado de multiprogramación es muy alto.

Para reducir el grado de multiprogramación, lo que se debe realizar es suspender procesos, es decir, enviar procesos a swap. Una vez más tenemos distintas alternativas, suspender:

- los procesos con baja prioridad
- los procesos que provocan muchos fallos de página: esto puede deberse a que el conjunto residente del proceso no sea el adecuado.
- el proceso activado hace más tiempo.
- el proceso con el conjunto residente de menor tamaño: esta estrategia se sustenta en el hecho de que el esfuerzo para volver a cargar el proceso será mínimo.
- el proceso con mayor cantidad de marcos: ésta estrategia busca obtener la mayor cantidad de marcos libres en memoria principal.
- el proceso con la mayor ventana de ejecución restante: ésta estrategia se aproxima al algoritmo de planificación "*primero el proceso con menor tiempo de ejecución"*

Unidad IX

~ Gestión de Entrada Salida ~

“Los discos duros me han decepcionado
más que la mayoría de las tecnologías.”

~ Steve Wozniak

Gestión de Entrada Salida

Introducción

Probablemente la E/S es el aspecto más complicado en el diseño de un sistema operativo y esto se debe a la gran variedad de dispositivos (Discos, teclados, mouse, pantalla, etc) que hace difícil establecer una solución general y uniforme.

Evolución del sistema de E/S

Conforme se fueron dando los avances tanto en los componentes como en los sistemas operativos, la complejidad y sofisticación de estos componentes individuales se fue incrementando. Estos avances y mejoras se pueden resumir en las siguientes etapas:

Técnica	Etapa	Descripción
E/S Programada	1era	El procesador controla directamente un dispositivo de E/S y es el encargado de mover los datos desde y hacia la memoria principal.
	2da	Se implementan interfaces en los dispositivos de E/S, lo que permite que el procesador se pueda independizar de los detalles de cada dispositivo. El procesador espera que se complete la operación de E/S y luego mueve los datos desde/hacia la memoria principal.
Uso de interrupciones	3era	Con el uso de interrupciones se logra que el procesador sólo participe al comienzo y al final de la operación de E/S, posibilitando la multiprogramación ya que el procesador puede ejecutar otro proceso mientras se realiza la transferencia.
DMA	4ta	A los módulos de E/S se les otorga el control directo de la memoria principal para que puedan mover un bloque de datos desde/hacia la memoria principal sin intervención del procesador. El procesador es interrumpido sólo al principio y al final de la operación.
	5ta y 6ta	Se mejora el módulo de DMA hasta convertirlo en un procesador independiente con su propio conjunto de instrucciones. El módulo de DMA puede ahora ejecutar procesos del SO para realizar la E/S.

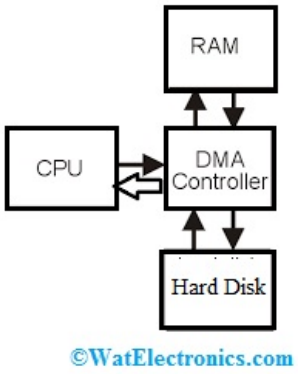
Nota: tanto la 1er como la 2da etapa “adolecen” de los que se llama espera activa ya que el proceso en ejecución debe esperar a que se transfieran los datos.

DMA

DMA o acceso directo a memoria (por sus siglas en inglés) es una tecnología que mejora la eficiencia de la transferencia de datos, que resulta muy conveniente cuando se requiere transferir grandes volúmenes de datos o simplemente para mejorar la eficiencia del CPU ya que se libera al mismo de la tarea de transferencia.

Cabe destacar que para implementar esta tecnología, se requiere de un módulo de hardware específico llamado DMAC (DMA controller o controlador de DMA). El DMAC suele estar integrado en el chipset de la computadora, que incluye los chips *northbridge* y *southbridge*. Dependiendo del diseño del sistema, el DMAC puede formar parte del chip northbridge, del chip southbridge o ser un chip (independiente) dedicado. En algunos chipsets modernos, este módulo puede estar integrado directamente en la CPU.

Un esquema simplificado de esta tecnología sería el siguiente:



¿Cómo funciona el acceso directo a memoria (DMA)?

Paso 1. Inicio

El inicio de la transferencia puede ser disparado por un proceso o bien por un dispositivo. Un proceso utilizará un *system call* para notificar al sistema operativo que necesita enviar o recibir datos desde un dispositivo.

Cuando un dispositivo necesita enviar o recibir datos de la memoria, inicia una solicitud DMA.

Una vez que se acepta una solicitud, el controlador DMA toma control temporalmente del bus del sistema para facilitar la comunicación directa entre dispositivos y la memoria.

Paso 2. Solicitud

Cuando un dispositivo necesita transferir datos desde o hacia la memoria principal, envía una señal de solicitud DMA al controlador DMA. Esta señal indica que la CPU no es necesaria para esta operación particular de transferencia de datos.

Al recibir la solicitud DMA, el controlador DMA verifica si el bus está disponible y luego inicia el acceso a la memoria. Al manejar estas solicitudes independientemente de la CPU, se reduce significativamente la sobrecarga del procesador y se aceleran las transferencias de datos entre dispositivos y memoria.

El controlador DMA gestiona la temporización y la priorización de estas solicitudes mediante técnicas de arbitraje eficientes. Esto asegura que múltiples dispositivos puedan comunicarse con la memoria sin problemas, sin causar conflictos o cuellos de botella en el flujo de datos.

Paso 3: Arbitraje

Cuando varios dispositivos necesitan acceder a la memoria simultáneamente, el controlador DMA arbitra entre estas solicitudes para garantizar una utilización eficiente de los recursos del sistema.

A través del arbitraje, el controlador DMA prioriza y programa tareas de transferencia de datos en función de reglas o algoritmos predefinidos. Este proceso ayuda a prevenir conflictos y asegura que cada dispositivo tenga un acceso justo al bus de memoria sin causar cuellos de botella o retrasos en las transferencias de datos.

Paso 4: Acceso a la memoria

Una vez que el controlador DMA obtiene el control del bus del sistema, puede acceder directamente a la memoria sin involucrar a la CPU. Esta interacción permite transferencias de datos eficientes y rápidas entre periféricos y la memoria.

Los datos se leen o escriben en direcciones de memoria específicas según las instrucciones que recibió el controlador DMA. El controlador asegura que los datos se transfieran con precisión y rapidez sin requerir la intervención constante de la CPU.

Paso 6: Transferencia de datos

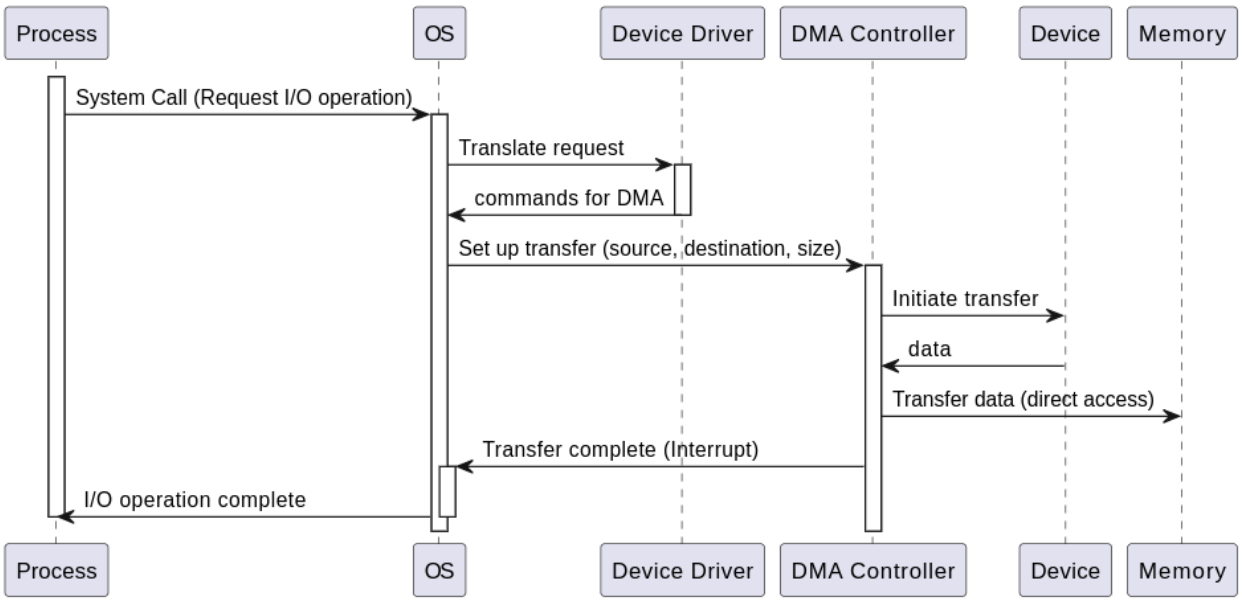
Una vez que el controlador DMA tiene control del bus, inicia el movimiento real de datos entre dispositivos y memoria. El controlador DMA se coordina con los dispositivos de origen y destino para transferir datos de manera eficiente sin involucrar a la CPU. Durante la transferencia de datos, la información fluye directamente de un dispositivo a otro a través de canales DMA sin intervención de la CPU.

Paso 7: Finalización e interrupción

Una vez completada la transferencia de datos, el controlador DMA activa una interrupción para notificar a la CPU. Esta interrupción indica que la operación DMA ha finalizado correctamente. Luego, la CPU puede reanudar sus tareas o manejar cualquier acción de seguimiento necesaria basada en la finalización de la transferencia de datos.

Al recibir una interrupción del controlador DMA, la CPU puede ejecutar rutinas o procedimientos específicos configurados para manejar tareas posteriores a DMA, como actualizar indicadores de estado, notificar a los componentes de software sobre la disponibilidad de datos o iniciar pasos de procesamiento adicionales basados en los datos transferidos.

A continuación se presenta un diagrama de secuencia simplificado de estos pasos:



Uso de buffers

Además de las complejidades que presenta manejar un dispositivo de E/S, este tipo de operaciones conlleva una serie de cuestiones de diseño del sistema operativo que permitan garantizar cierta eficiencia e imponer la menor sobrecarga posible. Imaginemos que un proceso requiere transferir datos desde el disco hacia la memoria principal. En ese punto el proceso debe esperar a que los datos estén disponibles para continuar su ejecución. Esta espera puede ser activa o utilizando interrupciones.

Esta situación presenta 2 problemas:

- el proceso debe esperar por la operación de E/S
- Si el sistema operativo decide suspender el proceso, esto interfiere con la política de reemplazo (ver capítulo de memoria virtual)

El 2do punto se debe a que el sistema operativo no puede reemplazar todas las páginas del proceso suspendido, al menos la página donde se van a almacenar los datos debe permanecer en memoria, de no ser así se pueden perder los datos o producirse un caso de interbloqueo de un único proceso.

Si el proceso lanza una operación de E/S y se suspende, el proceso se bloquea esperando el evento de E/S y la operación de E/S se bloquea esperando a que el proceso se cargue en memoria. Para evitar esta situación es que la página del proceso que se utilizará para recibir los datos debe permanecer en memoria. La misma situación se produce en caso de que la operación sea de salida y no de entrada de datos.

Para evitar estos problemas, es conveniente que se implementen las operaciones de E/S con buffers.

El uso de buffers es una técnica que si bien no permitirá que un dispositivo de E/S funcione a la velocidad del procesador o de la memoria principal, amortigua los picos en la demanda de E/S. En un entorno multiprogramado, el uso de buffers es una técnica que puede incrementar la eficiencia del sistema operativo y el rendimiento de los procesos individuales.

Buffer único

Para dispositivos orientados a bloques de datos, como los discos, las transferencias de entrada utilizan el buffer del sistema. Cuando se completa la transferencia, el procesador mueve el bloque de datos al espacio de usuario y pide otro bloque. Este proceso se lo denomina lectura anticipada ya que el bloque de datos se solicita antes de que el proceso lo requiera, con lo que el proceso puede estar procesando datos mientras se lee el siguiente bloque de datos.

Esta técnica si bien permite que el sistema operativo pueda expulsar el proceso de memoria, complica la lógica del sistema operativo ya que debe mantener un registro de los buffers asignados a los procesos y por otro lado, complica la lógica de intercambio ya que si el intercambio se realiza sobre el mismo disco no tiene sentido encolar la escritura del proceso en disco. Probablemente cuando la operación de E/S finalice ya no tenga sentido expulsar el proceso.

Las mismas consideraciones aplican a operaciones de salida de bloques.

Si el dispositivo es orientado a flujo de caracteres, como una pantalla o un teclado, el esquema de buffer único se puede utilizar en modo línea a línea o byte a byte, en vez de bloques de datos.

En el caso de E/S línea a línea, el buffer es utilizado para almacenar una única línea. El proceso de usuario se suspende durante la entrada de datos, esperando la llegada de la línea completa.

Con respecto a la salida, el proceso de usuario copia la línea de salida en el buffer y continúa el procesamiento. No se necesita suspender el proceso a menos que se tenga una segunda línea y el buffer no esté vacío.

Buffer doble

Con este otro esquema, un proceso transfiere datos a o desde un buffer mientras el sistema operativo vacía o llena el otro.

El uso de buffer doble debería “suavizar” el flujo de datos entre un dispositivo de E/S y un proceso, pero si el interés está en el rendimiento de un determinado proceso, sería deseable que el dispositivo pueda sostener el ritmo del proceso. La técnica de doble buffer podría no ser la más adecuada y habría que utilizar más buffers.

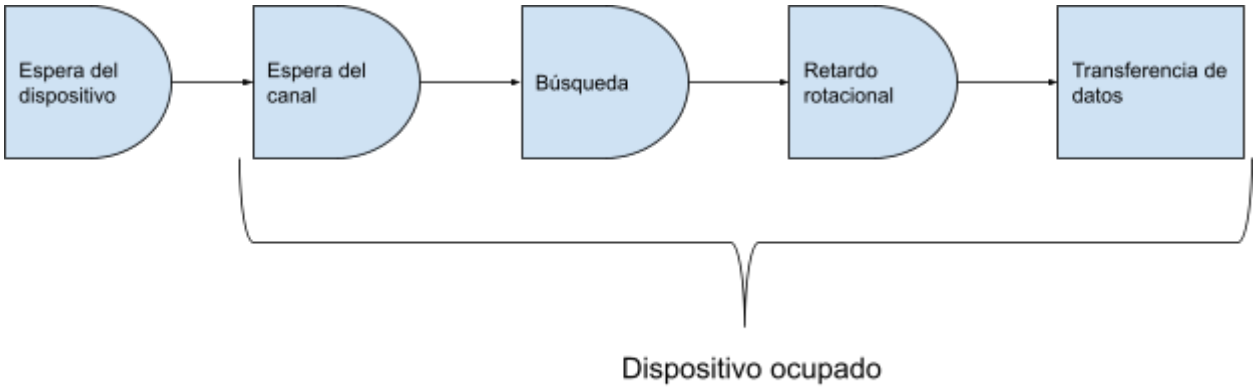
Buffer circular

Cuando se utilizan más de 2 buffers, el conjunto de buffers se los denomina buffers circular.

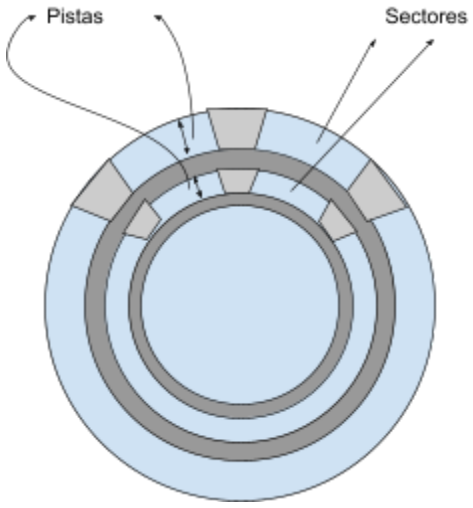
Planificación de disco

La planificación de uso del disco es de vital importancia ya que los mismos operan a una velocidad que es varias veces inferior a la velocidad de la memoria principal.

El tiempo de una transferencia depende del sistema operativo y del hardware involucrado (velocidad del bus, controlador de disco, el disco, etc). Un esquema general de los tiempos de transferencia de una operación de E/S de disco se puede representar como sigue:



Cuando el disco está en funcionamiento, se encuentra rotando a velocidad constante. Para leer o escribir, la cabeza se debe posicionar sobre la pista deseada y en el principio del sector. La selección de la pista implica el movimiento de la cabeza en un sistema de cabeza móvil o la selección de manera electrónica de una cabeza fija. En un sistema de cabeza móvil, el tiempo que se tarda en situar la cabeza en la pista se denomina **tiempo de búsqueda**. El tiempo que tarda en llegar al comienzo del sector se llama **retardo rotacional**, o latencia rotacional. La suma del tiempo de búsqueda, en caso de que lo haya, y el retardo rotacional se denomina **tiempo de acceso**, que es el tiempo que se tarda en llegar a la posición donde se debe leer o escribir. Una vez que la cabeza se posicionó, a medida que el sector se desplaza debajo de la cabeza se realiza la lectura o escritura. El tiempo que lleva realizar esta última operación se denomina **tiempo de transferencia**.



Políticas de planificación

El sistema operativo para procesar los pedidos de E/S, mantiene una cola por cada dispositivo. Los pedidos deben administrarse para que las operaciones de E/S no se realicen en orden aleatorio que sería el peor caso posible.

Los algoritmos que se describen a continuación se basan en la cantidad de movimientos que debe realizar la cabeza del disco para posicionarse en la pista solicitada.

FIFO

Es el algoritmo más sencillo para administrar las E/S. Si bien esta técnica tiene la ventaja de ser simple y equitativa, porque toda petición se acaba sirviendo, tiene un rendimiento bastante pobre en casos en los que existan muchos procesos compitiendo por el uso del disco.

Si son pocos los procesos que requieren acceso al disco y gran parte de las peticiones corresponden a sectores agrupados, entonces el rendimiento no es tan malo.

Prioridad

En un sistema basado en prioridad, la planificación del disco queda fuera del control del software de gestión de disco. Esta estrategia no se diseña para optimizar el uso del disco sino de los procesos. Se le otorga mayor prioridad a los procesos por lotes de corta duración y a los trabajos interactivos que a los trabajos largos que requieren más procesamiento. Esto permite que muchos trabajos cortos salgan del sistema rápidamente y proporciona buenos tiempos de

respuesta en trabajos interactivos. La desventaja es que los trabajos largos deben esperar, y hasta sufrir *starvation* en caso de que se agreguen al sistema procesos con mayor prioridad.

SSTF

Primero el de tiempo de servicio más corto, esta política consiste en seleccionar la petición de E/S del disco que requiera el menor movimiento del brazo desde su posición actual. De esta manera se busca que el tiempo de búsqueda sea mínimo. Este algoritmo proporciona un rendimiento mejor que FIFO. Como el brazo puede moverse en ambas direcciones, en los casos de empate se puede utilizar un algoritmo aleatorio.

SCAN

A excepción de la política FIFO, el resto de las políticas pueden dejar alguna petición sin servir. La política SCAN resuelve este problema (*starvation*). Con esta política, el brazo sólo debe moverse en 1 dirección satisfaciendo todas las peticiones que encuentre en su camino, hasta alcanzar la última pista en esa dirección o hasta que no haya más peticiones en esa dirección. En ese momento la dirección del brazo se invierte y la búsqueda continúa en la posición opuesta hasta agotar todas las peticiones.

Este algoritmo no aprovecha la proximidad como sí lo hace SSTF ya que una vez que pasó por una pista, deberá agotar todas las peticiones en esa dirección y recién cuando haga el barrido en la dirección opuesta atenderá a los pedidos que se encontraban cercanos a la pista recientemente dejada.

C-SCAN

C-SCAN o Circular SCAN restringe la búsqueda a una sola dirección , por lo tanto, luego de visitar la última pista en una dirección, el brazo vuelve al extremo opuesto del disco. Esto permite reducir el retardo máximo que pueden experimentar las nuevas peticiones con SCAN.

SCAN-N y FSCAN

Con SSFT, SCAN y C-SCAN es posible que el brazo no pueda moverse durante un período de tiempo considerable si existen procesos que tienen elevadas tasas de acceso a una determinada pista.

SCAN-N divide la cola de peticiones del disco en varias colas de longitud N. En cada momento se procesa sólo 1 cola utilizando SCAN. Mientras se procesa una cola, las nuevas peticiones se añaden a otra cola.

FSCAN aplica la misma política, sólo que maneja 2 colas únicamente.

Cache de disco

Una cache de disco es un buffer en memoria principal para almacenar sectores del disco. La cache contiene una copia de algunos de los sectores del disco. Cuando se hace una petición de E/S solicitando determinado sector, se comprueba si el sector está en la cache del disco. En caso afirmativo, se sirve la petición desde la cache. Debido al fenómeno de proximidad de referencias, cuando se lee un bloque de datos en la cache para satisfacer una única petición de E/S, es probable que haya referencias a ese mismo bloque en el futuro.

Consideraciones de diseño

El primer aspecto a considerar es cómo entregar los datos desde la cache a un proceso que los solicita. Se pueden abordar dos estrategias, la primera es copiar el bloque de datos desde la cache hasta la memoria asignada al proceso que lo solicita y la segunda y más performante es pasar un puntero al bloque de datos de la cache utilizando técnicas de memoria compartida.

El segundo aspecto de diseño está relacionado con la estrategia de reemplazo. Cuando se trae un nuevo sector a la cache de disco, se debe reemplazar uno de los bloques existentes.

El algoritmo más utilizado para este fin es LRU, se reemplaza el bloque que ha estado en la cache más tiempo sin ser accedido.

Otro algoritmo utilizado es el **menos frecuentemente usado (LFU)**, donde se reemplaza el bloque del conjunto que ha experimentado la menor cantidad de referencias. Cuando se trae un bloque a memoria, se le asigna un contador de referencias con el valor 1. Este contador se incrementa cada vez que el bloque recibe una referencia. Cuando se requiere un reemplazo, se selecciona el bloque con un contador más pequeño. Este algoritmo posee una deficiencia que es la siguiente: si las referencias a memoria se producen en intervalos de tiempo muy cortos, el contador de referencias no se incrementa, por lo que el principio de proximidad puede causar errores en el algoritmo LFU.

Una mejora a este algoritmo propone tener 3 pilas: nueva, intermedia y antigua donde se guarden las referencias a los bloques. Cuando se debe reemplazar un bloque, sólo se seleccionan bloques de la pila antigua. Esta modificación presenta mejores resultados que LRU y LFU.

RAID

Con el objetivo de seguir mejorando el rendimiento de la E/S, se crean distintas configuraciones de discos que proporcionan mejoras en cuanto a la velocidad de la lectura de datos, la

redundancia y la consistencia de los datos. Esto se logra con el desarrollo de vectores de discos independientes o RAID (redundant array of independent disk).

Existen 7 niveles o configuraciones, desde RAID 0 (cero) a RAID 6, que poseen las siguientes características:

1. Conjunto de unidades físicas de disco tratado por el sistema operativo como un único dispositivo lógico.
2. Los datos se distribuyen a lo largo de las unidades físicas de un vector
3. La capacidad de redundancia del disco se utiliza para almacenar información de paridad, que garantiza la recuperación de los datos en caso de falla.

RAID 0 y 1 carecen de la 3er característica.

RAID 0

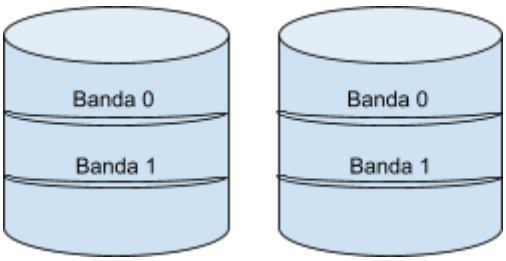
Este nivel no es un verdadero miembro de la familia RAID ya que no incluye redundancia.

En este esquema los discos se dividen en bandas (strips del inglés). Estas bandas se asignan de forma rotativa a cada disco.

La ventaja de este nivel es que los datos están distribuidos a lo largo de todos los discos del vector lo que facilita las lecturas en paralelo en el caso de que los datos solicitados no se encuentren en el mismo disco.

RAID 1

También llamado espejado (mirroring), logra redundancia duplicando los datos, a diferencia de los esquemas siguientes que lo logran aplicando cálculos de paridad. Proporciona una copia de respaldo (backup) en tiempo real.



Las principales características de este esquema son:

- ✓ Una petición de lectura puede servirse de cualquiera de los 2 discos que contienen los datos, aquel que proporcione el menor tiempo de búsqueda más la latencia rotacional.
- ✓ Proporciona tolerancia a fallos, ya que si un disco se rompe, todavía se pueden recuperar los datos del otro.

Las principales desventajas de RAID 1 son:

- ✘ El costo, ya que requiere del doble de espacio de disco. Siempre requiere una cantidad par de discos (2, 4, 6, etc).
- ✘ Una petición de escritura requiere actualizar ambas bandas. El rendimiento de la escritura lo impone el disco de mayor latencia rotacional.

RAID 2

En este esquema las bandas son muy pequeñas, pudiendo ser de un byte. RAID 2 , calcula un código de corrección de error con los bits correspondientes de cada disco de datos, almacenando los bits del código resultante en los correspondientes posiciones de bit en los múltiples discos de paridad. Normalmente se utiliza un Código de Hamming, que es capaz de corregir errores en un único bit y detectar errores en dos.

Si bien este esquema requiere menos discos que RAID 1, el costo es alto ya que requiere gran cantidad de discos redundantes. Otra desventaja es que en cada lectura se accede a todos los discos de manera simultánea y por cada lectura se verifica si hubo errores en los datos

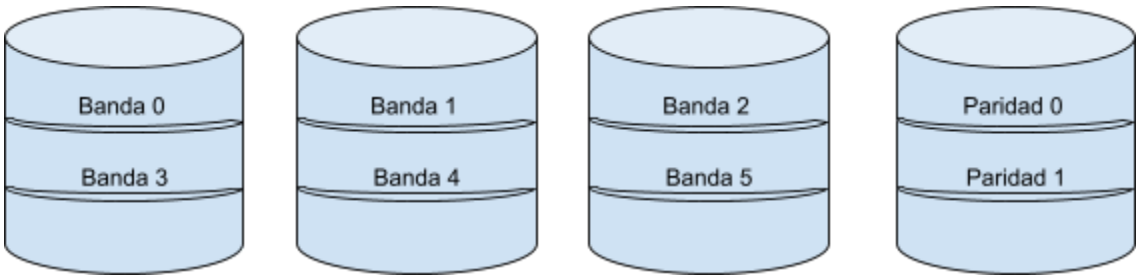
Dada la alta confiabilidad de los discos de hoy este esquema no es utilizado.

RAID 3

Similar a RAID 2, pero sólo requiere un único disco de paridad independientemente de la cantidad de discos de datos.

Este esquema guarda un bit de paridad mediante el cual puede detectar y reconstruir los datos en caso de que haya habido un error en la lectura. Esto permite reconstruir los datos de cualquier banda de datos. Este mismo esquema es aplicado en todos los esquemas siguientes.

Al igual que RAID 2 cada operación de lectura involucra todos los discos y sólo puede resolver 1 petición de lectura a la vez, por lo que no es recomendable para sistemas transaccionales.



RAID 4

Los discos de datos en este esquema son de acceso independiente, de forma que se pueden servir en paralelo peticiones de E/S independientes. Es por este motivo que RAID 4 es más adecuado para aplicaciones que requieran tasas de transferencia de datos bajas pero muchas peticiones de E/S. A diferencia de RAID 3 utiliza bandas relativamente grandes y también utiliza un único disco de paridad.

RAID 5

Es muy similar a RAID 4, la diferencia se encuentra en que las bandas de paridad se distribuyen en todos los discos. Ésta distribución evita los cuellos de botella sobre el disco de paridad.

Este esquema puede continuar funcionando a pesar del fallo de 1 de sus discos debido a que la información de paridad está distribuida en varios discos.

RAID 6

Este esquema extiende a RAID 5, generando 2 bloques de paridad en cada disco con lo cual permite el fallo de hasta 3 de sus discos y aún así seguir operando.

Este esquema utiliza 2 cálculos distintos para generar paridad.

Lectura recomendada:

- https://en.wikipedia.org/wiki/RAID

Bibliografía

- William Stallings. Sistemas operativos. Pearson Education. S.A., Madrid, 2005. ISBN-84-205-4462-0.
- Andrew S. Tanenbaum. Modern Operating System. Pearson Education Inc., 2009. ISBN-Q-IB-filBMST-L
- Multilevel Feedback queue. Wikipedia, La enciclopedia libre, 2019 [consulta: 21 de marzo del 2019]. Disponible en https://en.wikipedia.org/wiki/Multilevel_feedback_queue